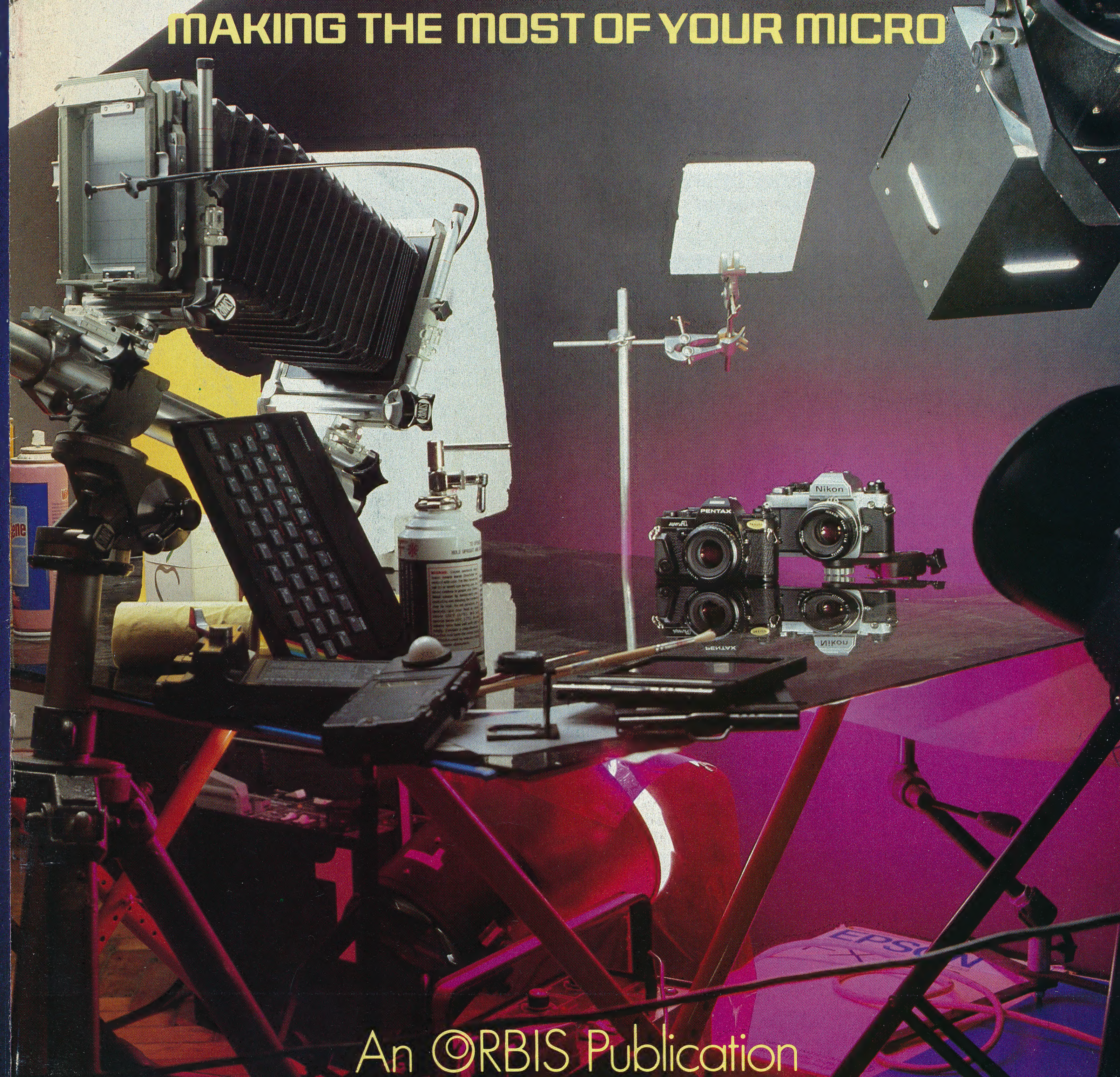
ISSN 0265-2919
80p 46

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

Next Week

- We take a look at the PC-1251 and PC-1500A — two pocket computers from Sharp.
- Continuing our examination of educational software, we examine the Macmillan series of Learn To Read packages for the ZX Spectrum.
- Communication between micros through the telephone network is becoming increasingly popular. We investigate how communication is achieved.

QUIZ

1) What kind of modem requires a telephone handset?
2) How much memory is contained on the chip used in the Canon T70?
3) Where would we need 'parity checking' and why?
4) What do we mean when we say that a procedure is state transparent?

QUIZ

Editor Mike Wesley; **Art Editor** Claudia Zeff; **Technical Editor** Brian Morris; **Production Editor** Catherine Cardwell; **Chief Sub Editor** Robert Pickering; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Staff Writer** Stephen Malone; **Sub Editor** Steve Mann; **Consultant Editor** Steve Colwill; **Contributors** Geoff Bains, Harvey Mellar, Jim Lennox, Soraya, Steve Colwill; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Controller** Peter Taylor-Medhurst; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; © **APSIF Copenhagen 1984;** © **Orbis Publishing Ltd 1984: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA** – Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders.
UK/EIRE – Price: 80p/IR£1. Subscription: 6 months: £23.92. 1 Year: £47.84. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Price: 80p. Subscription: 6 months air: £43.68. Surface: £34.84. 1 year air: £87.36. Surface: £69.68. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Price: 80p. Subscription: 6 months air: £45.76. Surface: £34.84. 1 year air: £91.52. Surface: £69.68. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Price: US/CAN$1.95/80p. Subscription: 6 months air: £54.08. Surface: £34.84. 1 year air: £108.16. Surface: £69.68. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Price: SA R1.95. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Price: 80p. Subscription: 6 months air: £58.24. Surface: £34.84. 1 year air £116.48. Surface: £69.68. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Price: Aus$1.95. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Price: NZ$2.25. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
ADDRESS FOR BINDERS AND BACK ISSUES – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 6711. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
NOTE – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
ADDRESS FOR SUBSCRIPTIONS – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited, Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER PHOTOGRAPHY BY IAN McKINNELL

TELEPHONE EXCHANGE

The Tone Of Command
Developed originally to connect remote terminals to mainframe computers, the modem ('MOdulator-DEModulator') converts, or modulates, digital electronic data into audio tones for telephone transmission, and demodulates audio tones back into digital signals on reception. Acoustic couplers transmit and receive via the telephone handset, while 'hard-wired' modems are connected directly to the line — usually through an extension socket

'Communications' is a term that covers any form of data transfer from one computer to another. In general, however, it's used to describe the process of sending information via the public telephone network. We begin a series of articles in which we explore in detail how the process works and the practical applications of this technology.

Connecting your micro to a modem allows it to talk to other computers over the telephone. A wide range of communications activities is then possible. You can send letters, which are received the instant they are transmitted, swap software with friends in distant places, exchange public messages with other computer users or gain access to a mainframe computer. We will discuss these applications in a later article. Here, we take a look at the principles behind data transmission between computers.

Communications technology (also known as 'comms') has its origins in mainframe computing. In the past, a typical mainframe configuration placed the computer in a purpose-built, air-conditioned room, from which connections were made with terminals scattered around the building. A terminal was simply a screen and keyboard connected to the main computer via a serial cable. Thus, a user could sit at a terminal at one end of a building and access the computer at the other end.

A direct serial link between the computer and terminal worked well over relatively short distances — that is, up to a few hundred metres — but signal deterioration ruled out longer distances, even if the cost of the cabling didn't. It was for this reason that the modem was developed.

A 'modem' — from '*mo*dulator/*dem*odulator' — is a device that allows computer data to be transmitted along an ordinary telephone line. It works by turning electrical signals from the computer into audio tones of a frequency and volume suitable for transmission through the telephone network. This process is known as 'modulation'. The receiving computer's modem then converts these audio tones back into electrical signals that can be passed to the receiving computer (this is 'demodulation'). A constant signal (called the 'carrier tone') is used as a reference, while the data is transmitted on the modulated wave.

The net result of this process is that it is possible to access remote computers almost as if they were directly connected to the terminal. Dedicated terminals usually consist of a VDU and a keyboard, and, because they have no processing power of their own, they are generally referred to as 'dumb terminals'. It is possible to use a microcomputer as a terminal, simply by using appropriate communications software. Because a micro has its own processing capability, however, it is known as an 'intelligent terminal'.

TYPES OF MODEM

There are two types of modem: 'acoustic' and 'hard-wired'. Acoustic modems — generally known as 'acoustic couplers' — have two rubber cups into which the telephone handset is placed. Audio sounds are transmitted through the mouthpiece and received from the earpiece. Acoustic modems are the more convenient of the two types, since they can be used with any telephone — and battery-powered ones can even be used with portable computers to make calls from payphones! On the other hand, acoustic modems are prone to interference from surrounding noise, and, therefore, can be unreliable.

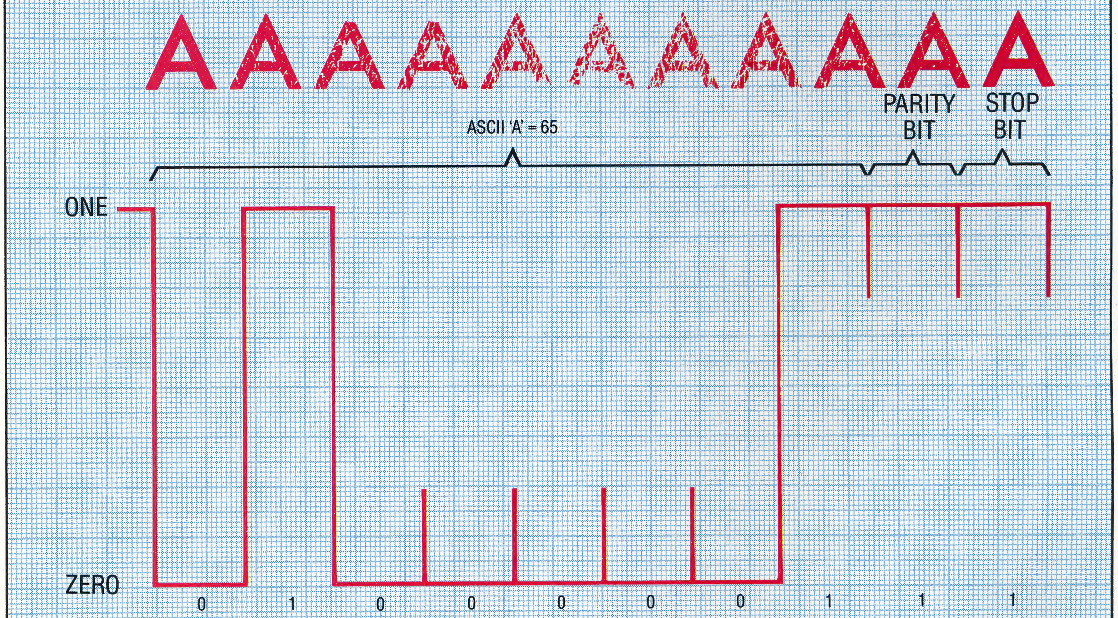
Hard-wired, or 'direct-connect', modems plug directly into a standard British Telecom jack socket, and the telephone itself plugs into the modem unit. As well as being more reliable than their acoustic equivalents, hard-wired modems generally offer more features.

Because hard-wired modems plug directly into the telephone network, they are required to be inspected by the British Approvals Board for Telecommunications, which is an organisation completely independent of British Telecom. BABT vets modems for safety, to ensure that there is no possibility of mains voltages being allowed to pass into the telephone system, and for efficiency, to make sure the modem disconnects 'cleanly' at the end of a call and does not leave the line 'hung'. Use of a non-approved modem is illegal, but this fact doesn't seem to have done much harm to the sales of those modems.

Useful additional features offered by some modems include 'auto-answering' and 'auto-dialling'. Auto-answer modems, upon answering the phone and detecting another modem at the other end, will pass control to the computer. If the call is not from another modem, the auto-answer modem will simply hang up. Auto-dial modems can accept a number from the computer and automatically dial it. Thus, in response to a name typed in by the user, the computer's software can look up the phone number in a database and then instruct the modem to dial that number.

A 'baud rate' is a measure of the speed of data transmission between two devices. It is normally

Train Of Thought



A Bit Of Parity
Data is transmitted over the telephone line by representing each character's binary ASCII code as a stream of audio tones. The frequency of the binary zero tone is just below that of the reference, or 'carrier', tone, while binary one has a tone frequency just above it. For error-trapping, extra tones — called 'stop' and 'parity' bits — may be generated: here, the transmitted character is 'A' — ASCII 65, or 01000001 binary. There are an even number of ones in this code, and the odd-parity system is in use, meaning that any code will always contain an odd number of ones; the parity bit is set to one, therefore. Following this is the stop bit, signalling the end of the character code. This eight-bit character code thus requires 10 bits in transmission

LIZ DIXON

thought of as the number of bits that are transmitted every second, although in practice this is not an accurate definition — for reasons we'll come to later in this article.

The two most common baud rates for communications devices are 300 and 1200/75 (the latter meaning that data is received at 1200 baud, and transmitted at 75). Most micros, by way of comparison, save programs to tape within the same range (between 300 and 1200 baud).

A 300 baud rate is used for systems where roughly equal amounts of data are being transmitted each way. These include bulletin boards and electronic mail systems like Telecom Gold. A 1200/75 rate is used for viewdata systems like Prestel, where most of the information is being sent in one direction.

Unfortunately, as with many other aspects of computing, these 'standards' are not universal. Because of differences between the UK and US telephone systems, the two countries use different frequencies. The UK frequency is known as the CCITT (or V21) standard, while the US equivalent is the Bell tone (named after the telephone company).

ALL ABOUT BITS

Because the telephone system can tell only a limited number of frequencies apart with any reliability, data is transmitted in binary form. To do this, each character is translated into its ASCII equivalent first, and then into binary form. Thus, the letter 'A' would be converted into 65 (ASCII) and then 01000001 (binary). Ones are represented by one frequency (just above the carrier tone, and thus referred to as 'high'), and zeros by another (just below the carrier tone, and thus referred to as 'low').

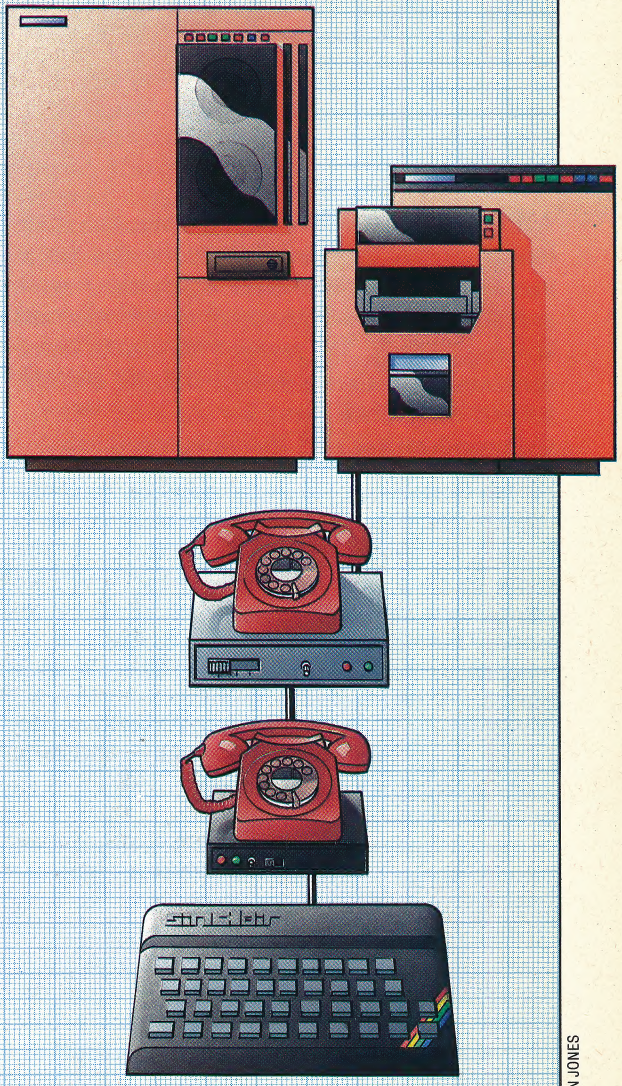
Because the receiving computer (known as the 'host') needs some way of telling where one character ends and the next begins, start and stop bits are also used. These are simply agreed frequencies; when the host receives a stop bit, for example, it decodes the preceding bits. The most common protocol for ASCII communications is eight data bits followed by one stop bit. An alternative setting is seven data bits and two stop bits. Other systems use both start and stop bits, the change indicating the end of a character.

Since communication over the public phone system is less than totally reliable, we need some form of error checking — a method by which the host can ensure that it has received a character correctly. The simplest solution is known as 'parity checking'. This involves counting the number of high bits and adding an additional parity bit to make the total number of high bits either odd (an 'odd parity' system) or even (an 'even parity' system). For example, returning to the letter 'A', the total number of high bits in 01000001 is two (an even number). Therefore, to make the parity odd, the parity bit would also have to be high. In the case of the letter 'C', however, which is translated as 01000011 in binary, the parity bit would have to be low.

Parity checking is relatively unsophisticated: it will detect an error in a single bit, but will not detect two errors in the same character since the parity would be correct. It is, however, simple and is useful for most applications where 100 per cent accuracy is not vital.

Earlier in this article, we mentioned that bits-per-second is not an accurate definition of baud rate. The reason for this is that start, stop and parity bits have to be taken into account. For example, in a parity-checked system using eight data bits and one stop bit for each character, only eight bits out of every 10 transmitted contain useful information. Thus, the true bits-per-second is 80 per cent of 300 baud — or 240 baud.

We have taken a detailed look here at the fundamental principles behind data transmission using modems. In the next instalment, we will consider further aspects of modem operation, such as duplex transmission and terminal protocols, and briefly discuss some of the uses to which computer communications can be applied.



Information Flow
Baud rate, named after JME Baudot, the French data transmission pioneer, is a measure of the speed of information flow between communicating devices. A baud rate of 1 means that information is being transferred at 1 bit per second. Typically, devices communicating via a modem use baud rates of 300, where roughly equal two-way transmission is required. Where data transfer is mainly in one direction as in viewdata-type communications, for example, a faster baud rate of 1,200 can be used. These transmission rates are very similar to the speed of data transmission during tape loading and saving

KEVIN JONES

SCREEN ROUTINES

Mode D'Emploi
In a BBC Micro program various 'trade-off' decisions must be made: hi-res modes use a lot of memory and support few colours; text modes use less memory, allow better colour ranges but support only medium or lo-res graphics. In this program, Mode 1 gives the necessary resolutions, but at the expense of a 20 Kbyte screen memory

Although most adventure games are text-based, some take advantage of the large memory and colourful graphics now available on home micros to create relevant screen displays. We present the first of three articles in which we design sample screens for our adventure game for the BBC Micro, Commodore 64 and the Spectrum.

In this instalment, we will consider how the graphics facilities of the BBC Micro can be used to create screen displays for adventure games. The game that we have been developing, which we have called Digitaya, is a text-based adventure game. That is to say, it uses words to describe the imaginary surroundings in which the player is placed. A text-based adventure, for example, would simply display the message 'You are in the throne room' to conjure up a setting, while a graphic adventure would attempt to draw a room with a throne.

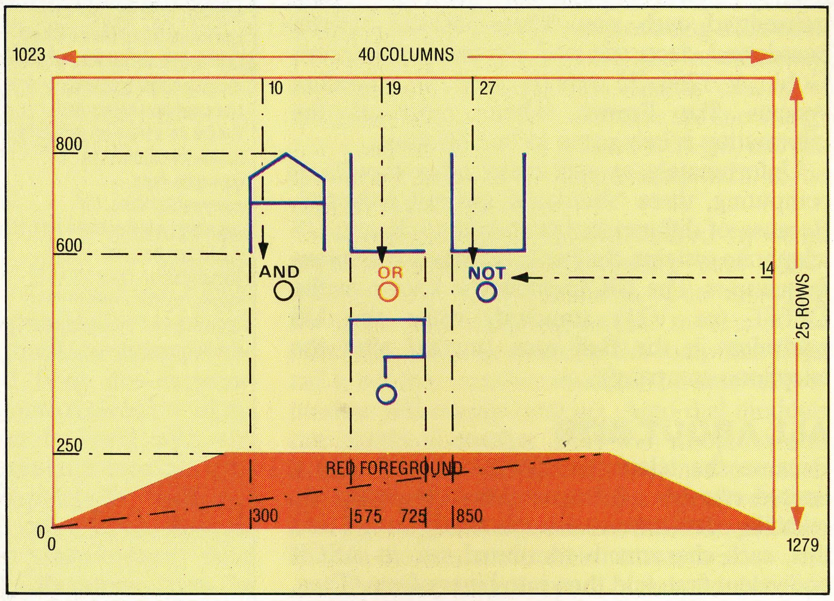
The screens that we will design here display two locations of particular interest in Digitaya: namely, the entrance to the joystick port and the Arithmetic and Logic Unit. The number of such screens is often limited by the amount of memory available; the commands required to produce each display take up memory space that would otherwise be available to increase the complexity of the plot.

ALU SCREEN DESIGN

Before we can start to design a screen for the BBC Micro, we must answer several questions:
1) How much memory do I have available?
2) How many colours do I need?
3) What standard of resolution is required?
All these questions can, in fact, be combined into one: 'What mode shall I use?'. Higher resolution and a wider range of colours mean that valuable RAM is taken up by the screen area. In our design, we shall use mode 1, which gives us four colours, a 40 by 25 screen and medium resolution. We should set the mode to be used by inserting the following line at the beginning of the program:

```
1095 MODE 1
```

Having decided on the mode, we can then sketch out what our screen is to look like, pencilling in suitable co-ordinates as we go. The design chosen here scrolls the upper-case letters A, L and U onto the screen. In the game, the player must press one of three buttons — marked AND, OR and NOT — and these must also be moved onto the display. Additional features include a thin border around the edge of the screen and a tapering foreground. Our rough design looks like this:

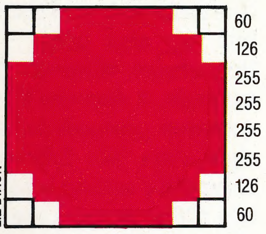


Each letter is formed by MOVEing to a start point and then using PLOT 1 to draw the shape of the letter as a series of lines relative to the start point. By designing the letters in this way they can be moved around the screen simply by changing the initial MOVE command. We can also rub out letters by redrawing the letter shape in the same position, but specifying Exclusive-OR plotting by using GCOL 3.

The buttons are formed by redefining a character. In this case, CHR$(240) is redefined by the procedure button to become the shape shown on the right. Notice that CHR$(240) is assigned to the variable button$ for use in the main part of the routine. The buttons and labels can be simply positioned by PRINTing them at co-ordinates specified by the TAB command.

The foreground is created using the triangular fill primitives provided by the PLOT 85 command. This command joins the point specified to the last two points previously plotted and then fills the resulting triangle with colour. The quadrilateral shape of the foreground can be drawn and filled by two such fill primitives.

The code for the screen display forms a subroutine of the special routine designed to deal with the ALU location in the game. The command A$=GET$, at line 7560, waits for a keypress before restoring the original foreground colour, clearing the screen and RETURNing to the main ALU routine to continue with the game. To call this graphic subroutine, the following line should also



On The Button
In the ALU picture for the BBC Micro a button shape is required to represent the three choices, AND, OR and NOT, available to the player. In the absence of a CIRCLE command or special PET-type graphics characters we must redefine an existing BBC character. Using an 8 by 8 grid we can design a shape and represent it using 8 decimal numbers. CHR$(240) can then be redefined using the VDU 23 command:

VDU 23, 240, 60, 126, 255, 255, 255, 255, 126, 60 be inserted in the main program:

```
4565 GOSUB 7000: REM ALU PICTURE S/R
```

JOYSTICK PORT SCREEN

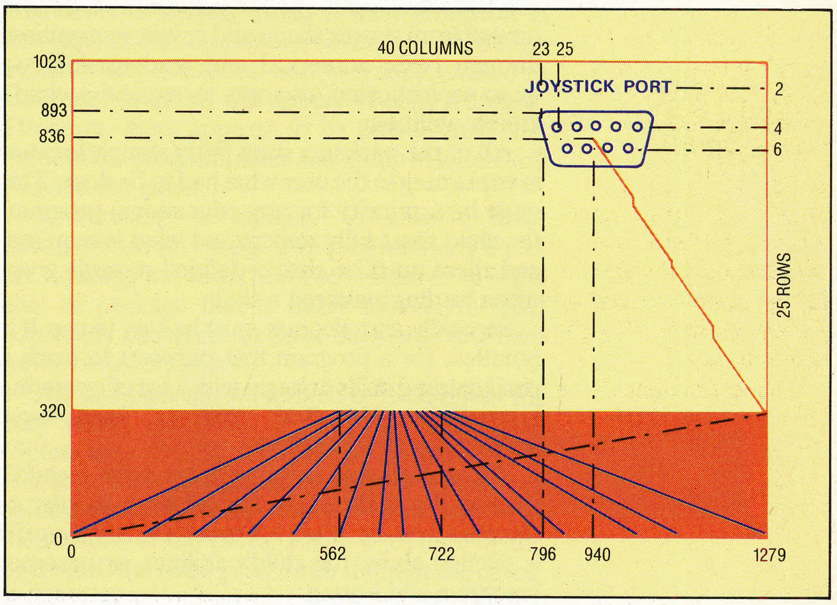
In Digitaya, if a player strays into the joystick port location, then he or she is in danger of being hit by a laser beam. The design of our screen display, therefore, involves drawing a joystick port with laser beams emanating from its centre. The joystick port is drawn using several full stop characters PRINTed to the top left corner of the screen, and a typical D-type surround is then drawn using high resolution graphics and PLOT statements. Notice that after MOVEing to the start position, all of the succeeding PLOT statements that create the port surround are PLOT 1 commands — which means they draw relative to the last point plotted. This is extremely convenient, because if shapes are drawn using a series of relative commands then, if it is decided to move the position of the whole shape, only the first MOVE statement has to be altered.

The foreground consists of a rectangular block of colour, once again drawn using two triangular fill primitives. To give an impression of depth, a series of converging lines is drawn over this, using a FOR . . . NEXT loop (lines 8170-8200). The loop sets up values of X from 0 to 1280 — the width of the screen in graphics units. A series of lines is drawn to the bottom of the screen, the start point on the horizon for each point increasing as X increases. However, the step of 32, used between consecutive lines at the bottom of the screen, is reduced to a step of 4 at the horizon (by dividing each X value by 8 in the MOVE command that defines the start point of each line).

The laser beam effect is produced by drawing a line from the centre of the joystick port to a randomly-chosen point on the horizon, in a random colour. The line is subsequently rubbed out — without disturbing the background — by plotting the same line in the Exclusive OR plotting mode, set by GCOL 3. The drawing and rubbing out of lines is placed within a REPEAT. . . UNTIL loop, together with a test to see if a key is pressed on the keyboard. Use of INKEY$, instead of GET$, allows program execution to continue while it is testing for a keypress within the loop. This loop is terminated when a key is pressed, the screen is then cleared, the original text colour restored and program control handed back to the main joystick port routine. To call this graphics subroutine the following line should be inserted:

```
3845 GOSUB 8000:REM JOYSTICK PORT PICTURE
```

Taking Some Stick
In both the joystick port and the ALU screens extensive use is made of the relative plotting facility since it permits easy erasure and movement of whole graphics shapes. Another plotting option is used to DRAW and FILL solid blocks of hi-res colour



ALU Screen

```
7000 REM **** ALU SCREEN S/R ****
7010 CLS .
7015 REM ** CURSOR OFF **
7017 VDU23,1,0;0;0;0;
7020 REM ** BORDER **
7030 GCOL 0,1
7040 MOVE 0,0
7050 DRAW 0,1023
7060 DRAW 1279,1023
7070 DRAW 1279,0
7080 DRAW 0,0
7090 :
7100 REM ** PATH **
7110 MOVE 1279,0
7120 PLOT 85,500,250
7130 PLOT 85,800,250
7140 REM ** LETTER A **
7150 GCOL 3,2
7160 FORX=0 TO 300 STEP 10
7170 FOR I=1 TO 2
7180 PROCletter_a
7190 NEXT I,^
7200 PROCletter_a
7210 :
7220 REM ** LETTER L **
7230 FOR Y=300 TO 590 STEP 10
7240  FOR I=1 TO 2
7250 PROCletter_l
7260 NEXT I,Y
7270 PROCletter_l
7280 :
7290 REM ** LETTER U **
7300 FOR X=1280 TO 850 STEP -10
7310 FOR I=1 TO 2
7320 PROCletter_u
7330 NEXT I,X
7340 PROCletter_u
7350 :
7360 REM ** BUTTONS **
7370  PROCbutton
7380 COLOUR3
7390  PRINT TAB(11,15)button$
7400 COLOUR1
7410 PRINT TAB(20,15)button$
7420 COLOUR2
7430 PRINT TAB(28,15)button$
7440 REM ** COMMANDS **
7450 COLOUR3
7460 PRINT TAB(10,14)"AND"
7470 COLOUR1
7480 PRINT TAB(19,14)"OR"
7490 COLOUR2
7500 PRINT TAB(27,14)"NOT"
7510 :
7520  MOVE 575,400
7530  PROCq_mark
7540 :
7550 REM ** WAIT FOR KEY **
7560 A$=GET$
7562 REM ** CURSOR ON **
7564 VDU23,1,1;0;0;0;
7566 COLOUR3:CLS
7570 RETURN
7580 :
7590 DEF PROCletter_a
7600 MOVE X,600
7610 PLOT 1,0,150
7620 PLOT 1,75,50
7630 PLOT 1,75,-50
7640 PLOT 1,0,-150
7650 PLOT 0,0,80
7660 PLOT 1,-150,0
7670 ENDPROC
7680 :
7690 DEF PROCletter_l
7700 MOVE 725,Y
7710 PLOT 1,-150,0
7720 PLOT 1,0,200
7730 ENDPROC
7740 :
7750  DEF PROCletter_u
7760 MOVE X,800
7770 PLOT 1,0,-200
7780 PLOT 1,150,0
7790 PLOT 1,0,200
7800 ENDPROC
7810 :
7820 DEF PROCbutton
7830 VDU 23,240,60,126,255,255,255,255,126,60
7840 button$=CHR$(240)
7850 ENDPROC
7860 :
7870 DEF PROCq_mark
7880 PLOT 1,0,60
7890 PLOT 1,150,0
7900 PLOT 1,0,-70
7910 PLOT 1,-75,0
7920 PLOT 1,0,-50
7930 PLOT 0,-8,-30
7940 PLOT 1,16,0
7950 PLOT 1,0,-16
7960 PLOT 1,-16,0
7970 PLOT 1,0,16
7980 ENDPROC
8000 REM **** JOYSTICK PORT PICTURE ****
8010 :
8020 REM ** CURSOR OFF **
8030 VDU23,1,0;0;0;0;
8040 CLS
8050 REM ** BORDER **
8060 GCOL 0,1
8070 MOVE 0,0
8080 DRAW 0,1023
8090 DRAW 1279,1023
8100 DRAW 1279,0
8110 DRAW 0,0
8120 :
8130 REM ** HORIZON **
8140 PLOT 85,1279,319
8150 PLOT 85,0,319
8160 GCOL0,2
8170 FOR X= 0 TO 1280 STEP 32
8180 MOVE 562+X/8,320
8190 DRAW X,0
8200 NEXT X
8210 :
8220 REM ** JOYSTICK **
8230 COLOUR 2
8240 PRINTTAB(23,2)"JOYSTICK PORT"
8250 PRINTTAB(25,4)". . . . ."
8260 PRINTTAB(25,6)" . . . ."
8270 GCOL 0,2
8280 MOVE 796,893
8290 PLOT 1,288,0
8300 PLOT 1,4,-4
8310 PLOT 1,4,-4
8320 PLOT 1,0,-4
8330 PLOT 1,-4,-4
8340 PLOT 1,-30,-81
8350 PLOT 1,-4,-4
8360 PLOT 1,-4,-4
8370 PLOT 1,-218,0
8380 PLOT 1,-4,4
8390 PLOT 1,-4,4
8400 PLOT 1,-30,81
8410 PLOT 1,-4,4
8420 PLOT 1,0,4
8430 PLOT 1,4,4
8440 :
8450 REM ** SHOOT **
8460 REPEAT
8470 A$=INKEY$(10)
8480 X=RND(1279):Y=320
8490 GCOL 3,RND(3)
8500 FOR I=1 TO 2
8510 MOVE 940,836
8520 DRAW X,Y
8530 NEXT I
8540 UNTIL A$<>"":REM WAIT FOR KEYPRESS
8550 :
8560 REM ** CURSOR BACK ON **
8570 VDU23,1,1;0;0;0;
8575 COLOUR3:CLS
```

START RIGHT

This is the first of two articles in which we look at a range of educational software written for young people. We begin with a brief discussion of the specific objectives a programmer must bear in mind when producing software for pre-school and primary school children.

The educational aims of the packages we looked at ranged from simple shape and colour recognition, through basic numerical and reading skills, to quite sophisticated attempts to expand a child's artistic abilities.

All of the packages were fairly straightforward in explaining to the user what had to be done. This must be a priority for any educational program: the child must fully understand what is required, and there must be clearly-defined rewards given when he has mastered a skill.

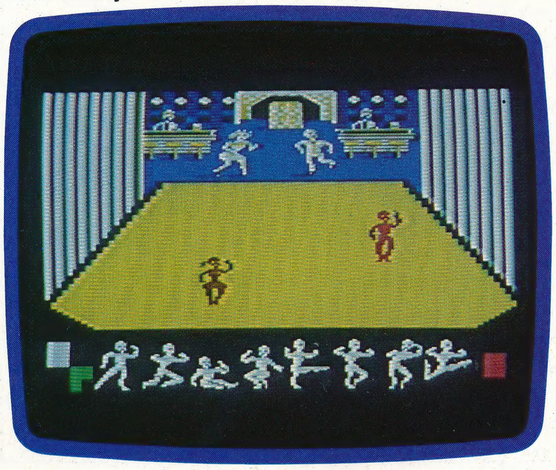
Secondly, the program must be easy to use. It is pointless for a program that purports to teach a child reading skills to begin with a list of operating instructions. The best programs keep these instructions to a minimum.

An educational program must hold a child's interest. No matter how important or worthy its ultimate aims are, it will fail to achieve anything if it is pitched above the child's abilities, or becomes repetitive and boring.

Finally, the acid test of an educational program is that it teaches what it is supposed to. This may seem an obvious point, but software houses often forget the educative aims of a program in favour of its entertainment value.

The packages we discuss here are produced by the American companies, Spinnaker and Fisher-Price. Although these programs are available in the US as cartridges, in the UK they are marketed in cassette format. Whereas a young child could easily be shown how to insert a cartridge into a computer and switch it on — thus being able to load his or her own program — the cassette format invariably requires an older person to be at hand to help load the program.

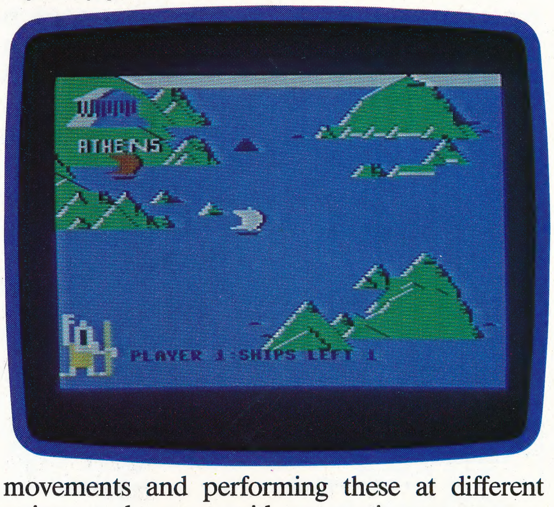
Dance Fantasy



Perhaps the most fascinating of all the programs we looked at, Dance Fantasy (Fisher-Price, £9.95) is aimed at four- to eight-year-olds. The screen display shows a stage on which two figures are standing, and the user is asked to choreograph a dance for them. Before beginning work you are asked to specify the sex of the dancers: a boy and a girl, or two boys or two girls.

At the bottom of the screen, the program displays a range of figures in various poses: each of these represents a particular dance routine — a leap, a jig, and so on. By moving one of the dancers over one of these figures, using the joystick, the child effectively chooses that particular routine, and then positions it on the stage and has it performed by pressing the fire button. A dance is thus choreographed by selecting a series of movements and performing these at different points on the stage, with connecting movements supplied by the program. Once a dance is complete, the child can SAVE it and then view the overall effect.

Aegean Voyage

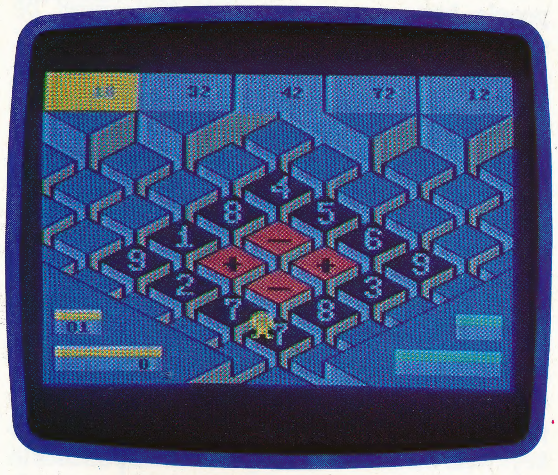


As you may have recognised from this description of Dance Fantasy, the program's great strength is that it is an imaginative analogy for a computer program: the child is able to create its own dance (program) using a series of basic routines (a set of procedures). This is then SAVEd to, and LOADed from, cassette — thus painlessly introducing the child to these two terms.

Aegean Voyage (Spinnaker, £9.95), aimed at a slightly older age group, uses characters and locations from Greek mythology as the elements of a simple adventure game. The objective is to sail a ship from Athens to various islands in the Aegean Sea, all the time avoiding rocks and storms. On reaching the safety of a port, the name of the island is displayed and a cryptic message appears at the bottom of the screen. The player must then decide whether to explore the island: a correct decision will be rewarded with treasure, such as the shield of Achilles; a wrong guess and the ship is sunk by a mythical creature, such as a Gorgon.

Classical scholars may be a little disturbed to discover that the game does get a few things wrong: the Minotaur, for example, is just as likely to appear on Delos as on Crete. The game makes no attempt to explain the significance of names or places: a child is hardly likely to gain even a rudimentary classical education from this cassette. It is unlikely that the game would hold a child's interest for long, since the graphics and format are uninteresting and repetitive.

Number Tumblers

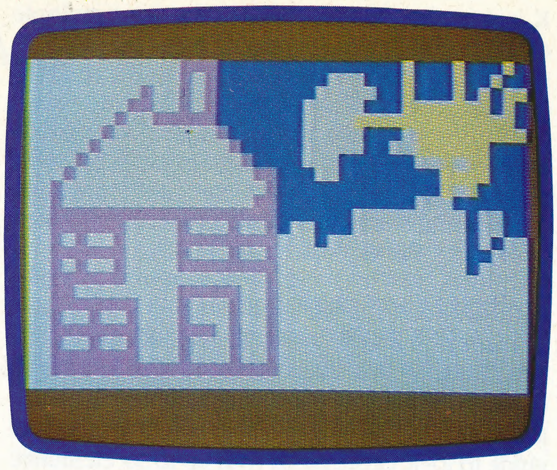


Designed to develop skills in mental arithmetic for eight- to 12-year-olds, Number Tumblers (Fisher-Price, £9.95) has the speed and feel of an arcade game. A series of numbers is displayed at the top of the screen, and the player has to arrange numerical and arithmetical symbols on the faces of a set of dice to create a mathematical expression that equals one of the numbers. The game is fast and enjoyable, has bright, well-designed graphics, and should provide a real incentive for a player to improve his or her mental arithmetic skills.

Kindercomp (Spinnaker, £9.95) is intended for children aged between three and eight. The aims of the package are to introduce young people to computers and develop artistic skills. The package consists of a series of different exercises for the child to use.

This package was originally written by Dr Doug Davis for his daughter, presumably as an entertainment. Unfortunately, Kindercomp gives the impression that it consists mostly of tricks with the computer — the sort of thing that most programmers design when learning BASIC and discovering the capabilities of the machine. For example, one of the options is 'Names'. The user is invited to input a name, or short sentence, up to 15 characters in length, which is then produced in assorted colours and sizes all over the screen. The effect is very attractive, and to a child unused to computer graphics will appear visually stunning. However, there appears to be little educational value in the program, since any group of letters will provide the effect.

Kindercomp

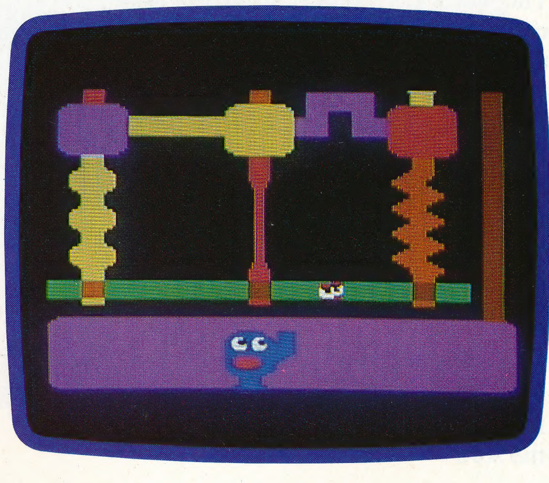


Kindercomp is unlikely to keep a child occupied for very long. The programming tricks are amusing but soon become repetitive. It is the sort of package that is likely to keep a child constantly occupied for three days or so and then never be used again.

The last package we looked at is designed for very young children. Alf In The Color Caves (Spinnaker, £9.95) features an amusing little character who slips and slides through a variety of colourful and variously-shaped tubes to a room at the bottom of the caves. Using a joystick or the keyboard, the child guides him through the caves, and if he is steered safely past a set of fast-moving ominous pairs of eyes, the user is rewarded with the spectacle of Alf doing a delightful little dance. Then he is sucked up a tube to ground-level again to begin another descent.

We have looked at only a few examples of the burgeoning range of educational programs aimed at youngsters. In the next instalment of the course, we will examine a further sample of currently-available packages.

Alf In The Color Caves



SCREEN SHOTS BY IAN McKINNELL

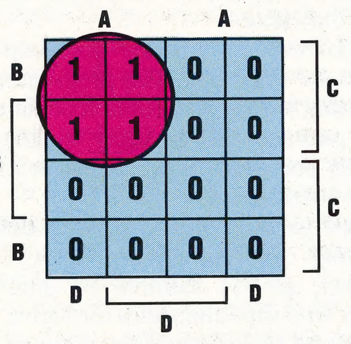
K

K

K is short for 'kilo-', a prefix indicating a multiple of 1,000. In computer terminology, however, *K* stands for a slightly different value — 2^{10}, or 1,024 decimal. Thus, 1 Kbyte equals 1,024 bytes; while 64 Kbytes equals 64 × 1,024, or 65,536 bytes. K by itself is commonly used to mean 'kilobytes', particularly in reference to a computer's memory. Thus, home computers are often advertised as possessing, for example, 64K RAM.

KARNAUGH MAP

A *Karnaugh map* is a graphic representation of a truth table in two dimensions. Similar to a Venn diagram, which shows how related sets intersect and overlap, Karnaugh maps provide a visual guide to logic expressions with up to six elements. The purpose of such a representation is to make clear how a Boolean statement, or other logical expression, can be simplified. This is then used to draw up circuit diagrams, comprising data lines and logic gates, which perform these procedures.



Map Making
Complicated Boolean expressions can be simplified by the logical operations of Boolean algebra, but this is often tedious and usually error-prone; a Karnaugh map (see page 92) gives an immediate picture of the expression and allows simplification by inspection. The map shown is that of the expression A AND C

KERNEL

The *kernel* is a set of commands that forms the nucleus of a computer's operating system. As such, the kernel is the lowest layer of the operating system — it must be present and it must be working properly for the system to function. Giving commands directly to the CPU, the kernel program makes sure that appropriate parts of the system are set aside to deal with instructions from the rest of the operating system program.

KEY

The word *key* has several meanings relevant to computer operation. The first is the most obvious: a key is a switch on a computer keyboard that, when pressed, sends a value to the CPU that is interpreted as a specific character.

In a database, a 'key' is one item of information that acts as a pointer for sorting. Information is stored in records, which are complete sets of data for individual items, sorted by field or category. In an address book database, for example, the name, address and telephone number of one person constitutes a record, where the data is stored in the name field, the address field and the telephone field. When sorting the database, you indicate the field that is to be sorted — this is called the 'key' for the sort.

Some database programs let you search on more than one field. In this case, the first level of the sort is called the 'primary key', and the next level the 'secondary key'. To complicate matters, some sophisticated database packages allow you to enter sentences, or even paragraphs, of descriptive text within a field. For the program to sort on that field, you must identify a key word or phrase within the text that it can search for.

A 'key' can also be the identifier that allows access to secure information — the term is derived from the expression 'lock and key'. A file is locked from unauthorised users and can be opened only with the proper key, which is usually a password or code number. This is especially important on networks or multi-user systems, where many people have access, but some files must be restricted.

Finally, an encryption code requires a 'key' on which messages are based and can be decoded. For example, in a substitution cipher, the key indicates the letters to be substituted for the actual message, or vice versa. Recent ciphers have used computer-generated random substitutions, which are virtually impossible to crack without the proper key.

KEYBOARD

A *keyboard*, simply, comprises a set of switches, each of which generates a unique signal when pressed. In the early days of computer systems, the keys were mechanical, using levers and punches to create holes in punch cards. Modern keyboards send signals electronically, and they vary from the membrane units used on the Sinclair Spectrum, to typewriter-style consoles with moulded plastic keys.

KEYPAD

A small keyboard with a specific function — such as entering data — is called a *keypad*. The most common form found on microcomputers is a numeric keypad, which has numeric and mathematical function keys in the same pattern as those on an electronic calculator or 10-key adding machine. The purpose of the keypad is to simplify fast entry of numbers for people who are accustomed to adding machines.

Numeric keypads can be incorporated into an alphanumeric keyboard by providing certain keys with a dual function — as in the Tandy Model 100. In this case, the letter keys M, J, K, L, U, I and O double-up with the values of the digits 0 to 6. These keys lie directly below the 7,8 and 9 keys on the top row of the keyboard, and the 10-key group resembles a numeric pad. Microcomputers like the ACT Apricot and IBM PC, which are used extensively in business, have a separate pad incorporated into the main console and located just to the right of the alphabetic keys. The Apple II range has no keypad built in, but one can be connected through the joystick port.

IN CAMERA

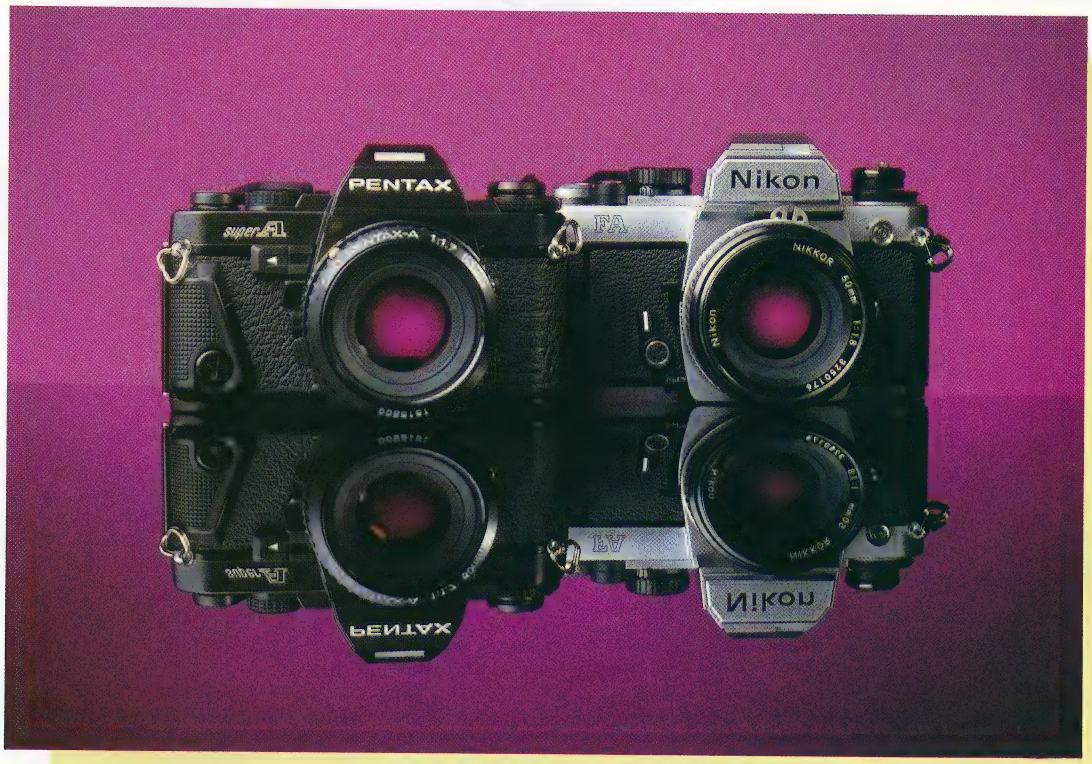
A major problem for the novice photographer is getting bogged down with the technical details of the equipment. The latest generation of cameras, however, has begun using the power of microprocessors to make picture-taking much simpler. We look at some of the processor-controlled cameras currently available.

When taking a picture, the photographer's first task is to decide on the correct *exposure*. This involves determining how much light from a particular scene will reach the film in the camera: too much and the picture will be bleached out, too little and it will be unviewably dark. To achieve the correct exposure, a suitable balance needs to be found between the setting for the *aperture* — the size of the gap in the lens, which determines how much light is let through — and the *shutter speed*, which determines the length of the exposure.

Therefore, the amount of light coming from a particular scene must first be measured and, taking into account the sensitivity of the film, the aperture and shutter speed set accordingly. Over the years, exposure meters were developed that allowed the photographer to take a reliable measurement of the brightness of a scene. More recently, exposure meters have been built into cameras, although the photographer still has to select a shutter speed and aperture setting to match the meter reading.

The growth of electronics in the 1970s has made it possible for the reading from the light meter to be translated directly into settings for the aperture or the shutter. This is done without any intervention from the photographer, so good quality results can be obtained simply by pointing a camera at a subject and shooting. This is a particularly useful facility for both the beginner, who may want to take photographs without understanding how cameras work, and the professional news photographer, who needs to snatch photographs under difficult conditions.

The Canon A1, priced at £260, allows six different modes for taking photographs. They are:
1) *Shutter priority:* the user picks a shutter speed and the camera sets the corresponding aperture.
2) *Aperture priority:* the user chooses an aperture and the camera sets the shutter speed.
3) *Program:* the camera sets both shutter speed and aperture using a program that produces an optimum combination of the two.
4) *Automatic flash:* when fitted with certain flashguns, the camera automatically sets the right shutter speed for flash (1/60 second) and sets the



IAN McKINNELL

Camera-Ready
Microprocessors have been controlling the interrelated functions of light metering and aperture-setting since the 1970s. Today's microprocessor-controlled cameras are capable of much more: taking care of shutter, exposure and flash functions to allow a novice user to produce high quality pictures, even in abnormal light conditions. Two such cameras are the Nikon FA and Pentax Super A, capable of running alternative control programs to cater for a range of different conditions

aperture to match the flashgun. A light meter in the flashgun cuts short the power of the flash when enough light has bounced back from the subject.
5) *Stopped-down aperture priority:* this is for old-style lenses and certain accessories that work with the lens aperture always 'stopped down' to the value used for taking the photograph. Modern lenses remain at the maximum aperture setting, other than at the instant the photograph is taken, so as to keep the viewfinder image as bright as possible.
6) *Manual:* both shutter speed and aperture are set by the photographer. This is useful for total control when a special effect or awkward lighting is involved.
An electronic display inside the Canon A1's viewfinder tells the photographer what mode the camera is in and what values it has set for the shutter and aperture. The display uses LEDs so that it can still be seen in the dark.

Despite the general usefulness of the program mode on the Canon A1, it works to a fairly simple formula. This means, in a few cases, it does not pick quite the best possible combination. For example, if photographs were being taken in the late evening, the camera might select a shutter speed of 1/30 second and an aperture of f/2.8. Any photographs taken at a shutter speed slower than 1/60 second risk being spoiled by the slight movements of the photographer's hand (known as 'camera shake'). The camera flashes a warning of the danger of camera shake when its shutter speed goes below 1/60 second, yet the program does not select a larger aperture to allow the faster shutter speed.

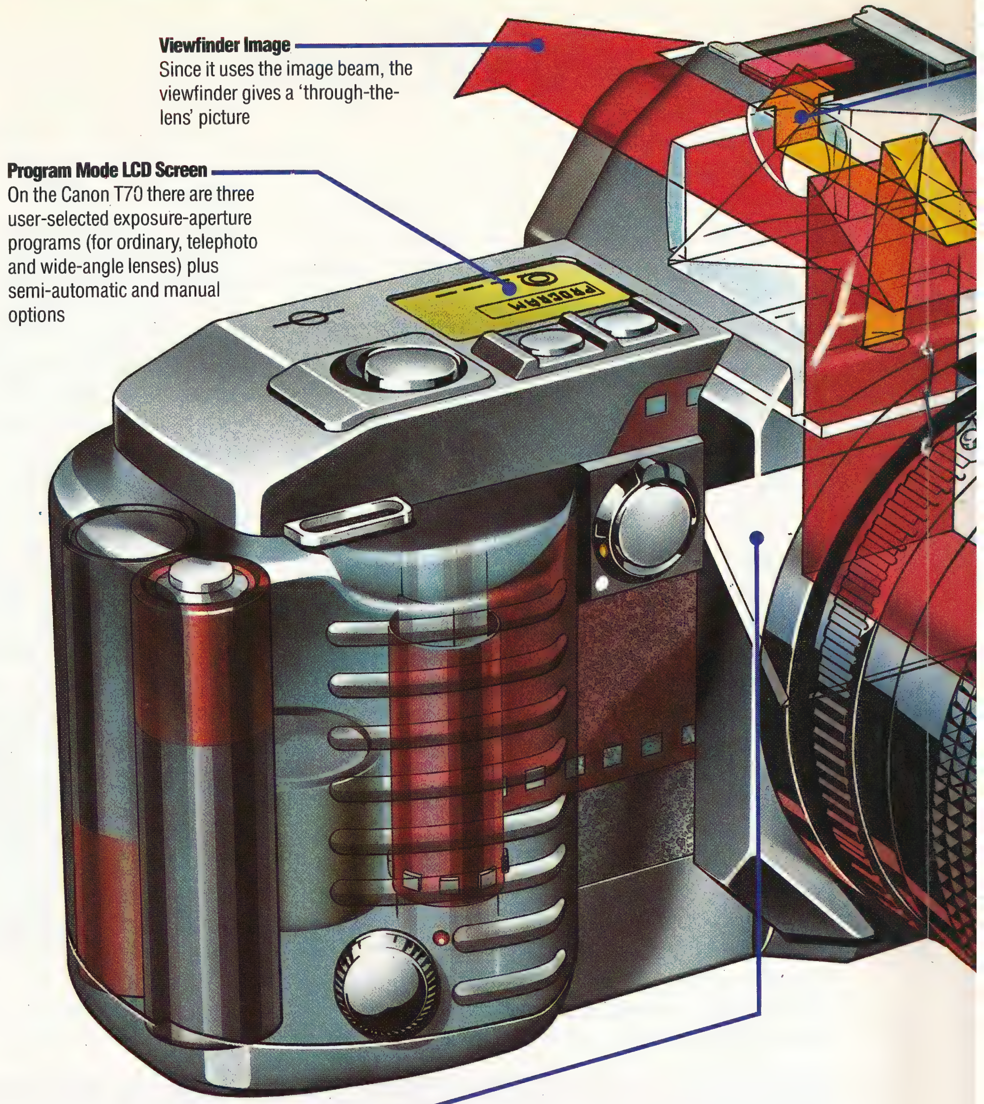
CANON'S COMPETITION

A number of rival companies have launched multi-mode cameras with built-in microprocessors. The Pentax Super A, retailing at £240, uses a slightly more sophisticated program than the Canon A1, which gives it a better combination of shutter speeds and apertures in both bright and dim light. In the 'late evening' situation we just described, the Pentax Super A would select a speed just below 1/60 second, so there would be less chance of camera shake. And the Nikon FA, manufactured by Canon's arch rival, automatically detects when a telephoto lens (135mm focal length or greater) is fitted, and accesses an alternative program. This is optimised to avoid camera shake with a longer lens by using faster shutter speeds and larger apertures.

Following the Nikon lead, Canon has used three alternative programs in its latest camera, the Canon T70. One is intended for ordinary lenses, one for telephoto lenses and the third for wide-angle lenses. However, the camera does not automatically recognise which lens is fitted, so the user has to select the appropriate program. This isn't necessarily a drawback, since it does allow a little extra creative control. For example, if you're shooting a fast moving subject with a wide-angle lens you can select the telephoto mode to get fast shutter speeds, and ensure that the action is frozen.

Another problem with automatic cameras is that they give an average exposure for the whole image, and the meter can be easily fooled by subjects with an extreme range of brightness in the image. For example, if a motorcycle is photographed against a sunset, the camera will tend to give the right exposure for the sun and make the bike much too dark. On the other hand, if the motorcycle were photographed against a black background, the camera would treat the subject as being much darker than it is and the photograph would probably be overexposed.

The Nikon FA, in a slightly higher price range at £410, uses a novel way to get around the problem. Instead of taking one measurement of the brightness of a scene, it measures five different parts of it. The FA then uses a microprocessor to compare the five readings with various 'standard

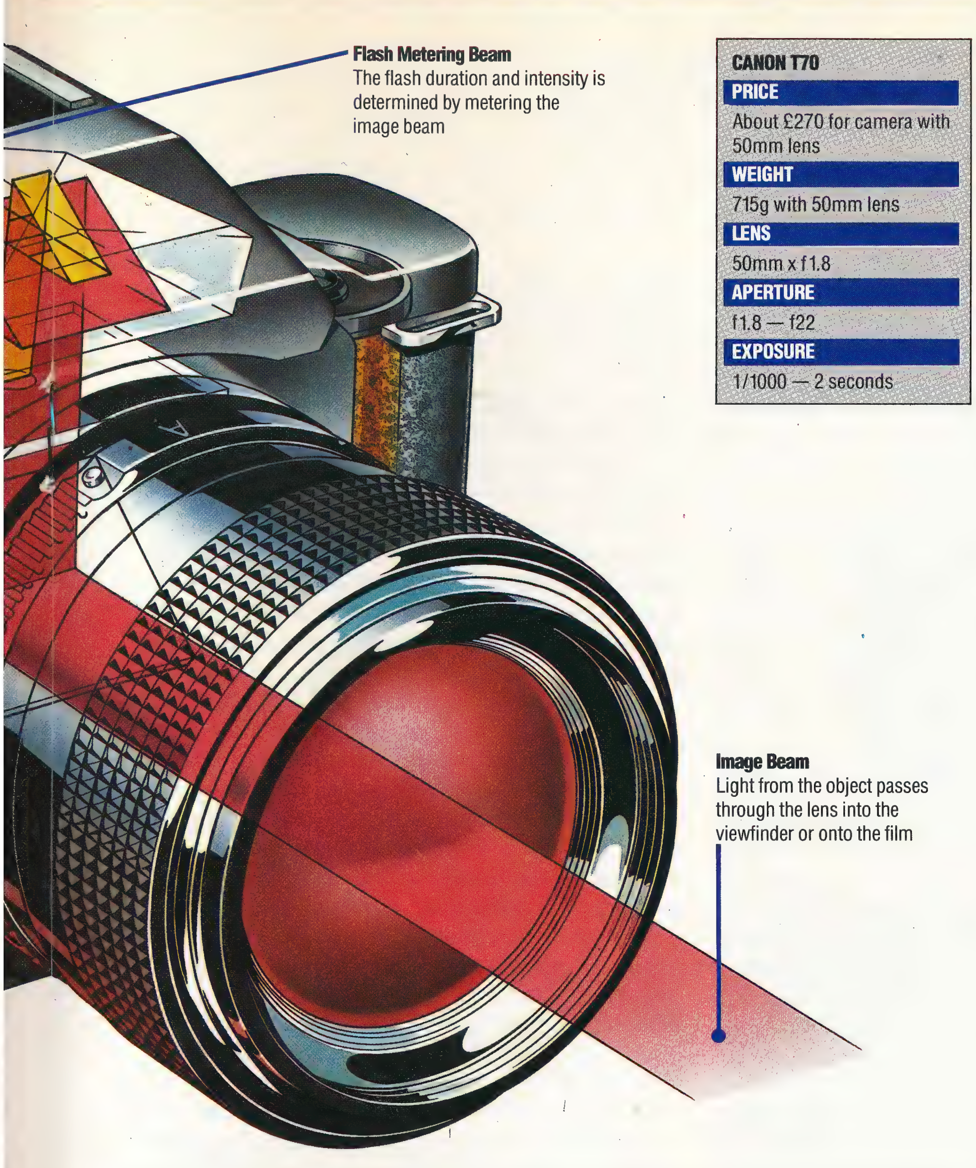


Viewfinder Image
Since it uses the image beam, the viewfinder gives a 'through-the-lens' picture

Program Mode LCD Screen
On the Canon T70 there are three user-selected exposure-aperture programs (for ordinary, telephoto and wide-angle lenses) plus semi-automatic and manual options

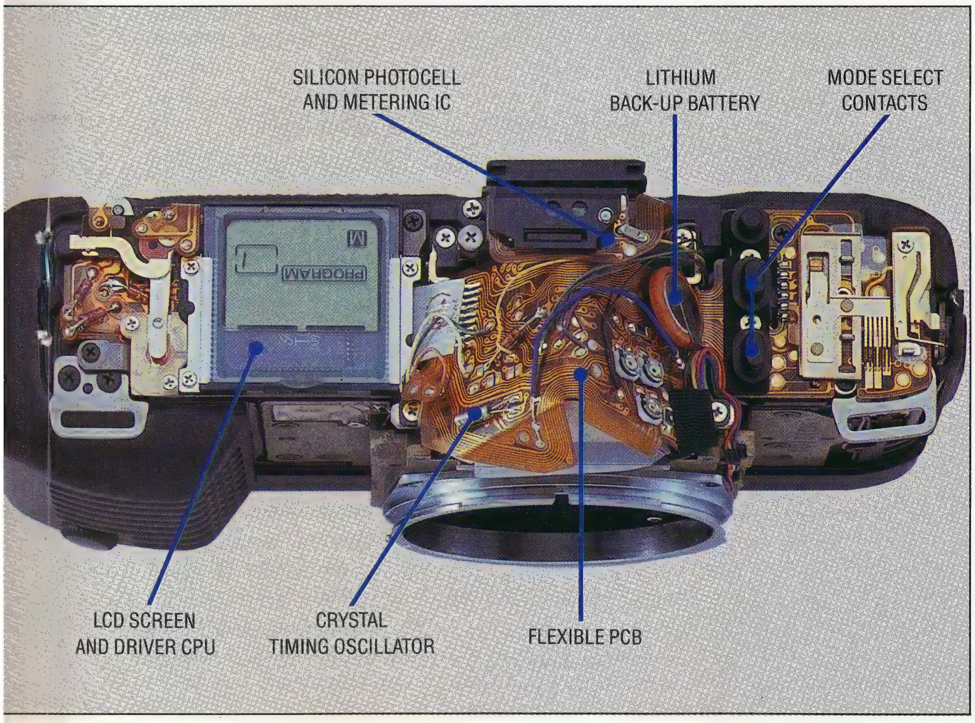
Sprung Mirror
Directs the image beam into the viewfinder prism until shutter release is pressed

Micro Photos
A purpose-built 8-bit CPU controls the overall operation of the Canon T70, assisted by metering and ISO chips. The crystal timing oscillator generates the clock synch pulses and controls the length of the exposure. Spring-loaded contacts controlled by the metering IC set the aperture. An optional command module can be attached which gives automatic interval exposure (between one second and one day) and allows timing data to be written directly onto the negative


Flash Metering Beam
The flash duration and intensity is determined by metering the image beam
Image Beam
Light from the object passes through the lens into the viewfinder or onto the film

CANON T70

PRICE	About £270 for camera with 50mm lens
WEIGHT	715g with 50mm lens
LENS	50mm x f1.8
APERTURE	f1.8 — f22
EXPOSURE	1/1000 — 2 seconds


SILICON PHOTOCELL AND METERING IC
LITHIUM BACK-UP BATTERY
MODE SELECT CONTACTS
PROGRAM
M
LCD SCREEN AND DRIVER CPU
CRYSTAL TIMING OSCILLATOR
FLEXIBLE PCB

scenes' programmed into the camera. Each of these scenes is produced by analysing thousands of photographs.

But of all the cameras currently available, the one that makes the fullest use of electronics is the Canon T70, which is priced at a very reasonable £240. The T70 has no mechanical controls; all of its settings are made by pressing buttons. One of eight modes can be selected by pressing a button on the top left of the camera. This makes information appear on a large LCD on the top right of the camera. Important information, such as shutter speed, aperture and mode is, also shown in the viewfinder, so that the photographer does not have to take the camera away from his or her eye while framing a shot.

CANON T70 IN OPERATION

When selecting a shutter speed, adjustments are made to the current value displayed on the LCD, by using two buttons to step the speed up or down. The film speed (the sensitivity of the film, known as its ASA or ISO number) is set in much the same way. The frame counter, showing how many photographs have been taken, also appears in the liquid crystal display.

The camera has a built-in motor to advance the film, and rewind it when the roll is finished. The T70 runs off two ordinary batteries, and three bars in the LCD register their status. If all three bars are shown, the batteries are fresh; two bars mean they are partly used; and one bar means they need replacing. If the camera's self-timer is used, a display on the LCD counts down the seconds until the shutter is released.

The microprocessors used in cameras are much less powerful than those used in computers. The T70 has its own specially-made eight-bit microprocessor, which is of the CMOS type to keep its power consumption at the minimum. It works at a clock speed of only 32 KHz; microcomputers work about 100 times faster. When not being used, the camera switches to a mere eight KHz to save power.

The microprocessor is encased in a 60-pin flat package and has a large amount of ROM, but only 16 bytes of RAM. Four other chips work with the microprocessor, the most important being the input/output chip. This controls the mechanical operation of the camera via magnets and a motor. It also converts the analogue electrical signal from the light meter chip into a digital signal so that the microprocessor can understand it.

The power of microelectronics makes it possible to get a great deal of pleasure from taking good quality photographs without needing to understand the complexities of photography. Even so, there will always be cases where the camera will give the wrong exposure or focus on the wrong object. The person who really understands the operation of a camera will always have the advantage over a beginner with a piece of sophisticated photographic equipment, but each year that advantage is getting smaller.

POWER SUPPLY

By connecting a servo motor to the user port of your computer, using the buffering system that we have developed in the Workshop course, it is possible to undertake a variety of control applications. We look at what servo motors are, and suggest some uses to which they can be put.

There are three types of electric motor — direct current, stepper and servo. A *direct current* (DC) motor can be easily controlled by a computer, but tends to be inaccurate if it encounters any resistance. In such a situation, the speed of the motor is reduced and the controlling computer cannot keep track of its position.

A *stepper* motor does not have this problem, as it moves through a fixed angle (for example, 7.5°) each time it is given a single current pulse. By counting pulses, and assuming that the motor is never overloaded, the computer can calculate the motor's position. Stepper motors are widely used in computer-controlled systems — robot arms, lathes, sorters, etc. However, the control pulses also deliver the power to the system, and the motors therefore require specially-built drivers and are relatively expensive.

Small digital *servo motors* are easily obtained from modelling shops, since they are commonly used in radio-controlled aircraft, boats, cars, and so on. These motors vary in size from about half that of a matchbox to almost ten times that size. Prices generally start at £8. Some servo motors are remarkably strong — capable of delivering more 'torque' (the word used to describe any force causing rotation) than most people could produce using a large screwdriver. Even the least expensive of these are suitable for making small robot arms, etc.

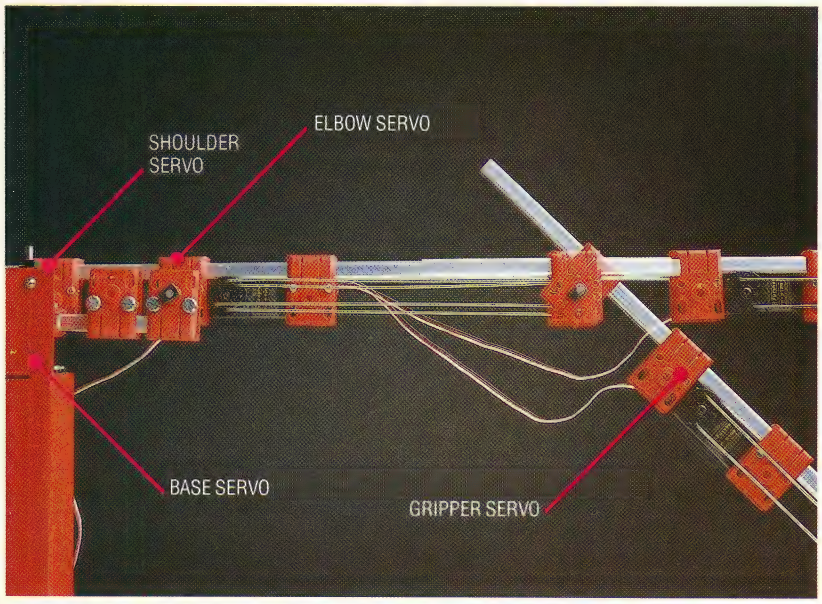
A small digital servo motor, such as a Futuba FP-S126 or an Axoms AS-1, contains a feedback potentiometer and a tiny DC motor linked — via a series of gears — to a 'horn'. The latter is a protrusion from the motor onto which levers, sprockets and so on can be attached. The motor is also ideal for setting up a looping feedback system with a computer, as the motor case also contains all the circuitry to do so, as well as an integrated circuit controller chip.

A typical model-maker's digital servo motor is powered from a five volt supply, and its angle (position) is set through a separate control wire. A one-millisecond (1/1000) pulse will move the horn in one direction, while a two-millisecond pulse will move it the same angle in the other direction. Variations in the angle moved are proportional to the pulse duration.

However, the motor will remain active only for about 20 milliseconds after the pulse, when it 'relaxes' and returns to its original position. To maintain the lever at a particular angle, therefore, the control pulse must be repeated at a frequency of about 50Hz.

Servo motors are normally used to move levers, and so on, but they can also be used for linear motion. If the potentiometer is decoupled from the gear train and centred, the motor will rotate continuously. In fact, the speed it rotates at is then governed by the pulse duration.

Beast Of Burden
The Beasty robot arm (see page 770) is powered by three servos (base, shoulder and elbow rotations) with an optional fourth servo to drive the end effector. The servo motor is ideal for robot arm use precisely because it can be moved and then locked in position



IAN McKINNELL

COMPUTER CONNECTION

When servo motors are directly connected to the user port of a computer, then wiring errors can damage the delicate internal mechanisms of the machine. Therefore, a buffering system must be used, and the buffer and output boxes built earlier in Workshop (beginning on page 514) are ideal for this. You should connect the motor's power supply wire to the positive socket (red) of one of the lines of the low voltage output box. The common ground wire should be connected to the negative socket (black).

If the motor control wire has been connected to data line 0 on the computer user port, then the motor itself can be controlled by sending the appropriate pulse sequences to the user port data 0 line. A pulse is sent by raising data 0 to five volts, by storing a binary 1 in data 0. A countdown loop is then used to wait for the desired length of time before lowering the output again by storing a binary 0 at data 0.

Both the BBC Micro and the Commodore 64 use versatile interface adaptor chips to form the user port. Since a user port can be used for both input and output, the port that we are using must first be set to the required mode (in this case, for output). In both micros, this can be done by POKEing the data direction control register directly using BASIC.

Now we must consider how to pulse a user port data line. If a value of hexadecimal 88 (equivalent to decimal 136, or a binary bit pattern of 10001000) were POKEd to the user port, the data pin voltages would be: 5v, 0v, 0v, 0v, 5v, 0v, 0v, 0v, respectively. This pattern would remain until it was deliberately altered. Therefore, a simple pulse can be generated on data line 0 by POKEing hex 00, hex 88, hex 00.

For the pulse to be fast enough, machine code must be used. The algorithm for pulsing a single servo motor is:

1) Specify the angle of the lever by storing it in a single byte (called ANGLE) with a value between 0 and 255, using a BASIC program.
2) Set data line 0 high (5v), thus starting the pulse.
3) Wait one millisecond by looping and decrementing the counter.
4) Wait a further period of between 0 and 1 millisecond, again by looping, but this time the starting value of the counter (and therefore the number of loops) is ANGLE.
5) Clear data line 0 low (0v) to end the pulse.

If ANGLE = 0, the pulse will last for a duration of one millisecond; if ANGLE = 128 (on the BBC Micro and Commodore 64) it will last 1.5 milliseconds, and the lever will move to a midpoint.

A STREAM OF PULSES

One pulse, however, is not enough to maintain the position of a servo motor. It must receive a stream of pulses, refreshing the motor about every 20 milliseconds. There are two ways to produce a stream of pulses:

1) By simply using a wait loop to pause between pulses. But this means that the computer can do nothing else while looping.
2) By using 'interrupts', which allow the computer to run another program — usually in BASIC — almost simultaneously. This background program can instruct the motors where to move.

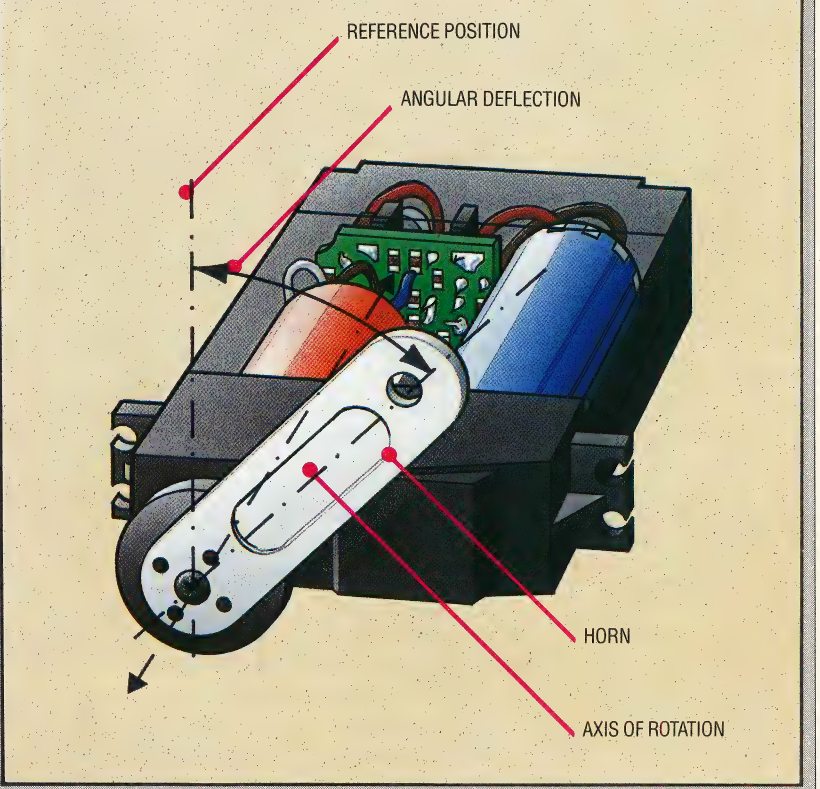
Both the BBC Micro and the Commodore 64 use 6500 series processors, which have two interrupt pins — NMI and IRQ. The second, the interrupt line, will be used for our timing tasks. Whenever a pulse appears on the IRQ line, the processor stops what it is doing and starts to execute the interrupt handling program. When that is completed, it returns (RTI) to where it left off when it was interrupted.

Both the BBC Micro and Commodore 64 use interrupts to run their operating systems. The BBC Micro generates 100 interrupts per second (one every 10 milliseconds) and the 64 has a rate of 60 per second. On each interrupt the system timers are updated, the keyboard scanned, and so on. Thus, the operating systems of these machines have clocks that generate interrupts, and also have handlers to trap and use them.

In both computers, the system interrupts can be used to run the pulse generating program. The Commodore 64 interrupts must be intercepted by changing the interrupt vector. This vector — a two-byte address held in two consecutive cells — tells the processor the location of the interrupt handling routine. By changing this address to point to the pulser routine, and directing the processor back to the usual system interrupt handler at the end of the pulse, the processor will generate a pulse whenever an interrupt occurs — namely 60 times a second.

We give here the BASIC and Assembly language versions of a program to control a single servo motor on the Commodore 64. In the next instalment of Workshop, we will give listings for control of several motors at once, and the equivalent listings for the BBC Micro.

They Also Serve . .



The deflection of the 'horn' attached to the servo's spindle is determined by the duration of the pulse sent by the controlling computer — the longer the pulse, the greater the deflection. If the pulse is repeated at intervals of about 20 milliseconds, the horn will be held in position by the motor — this 'position-locking' makes the servomotor ideal for electro-mechanical control applications. Most microcomputers generate interrupts every 10 to 20 milliseconds, so this 'refresh' signal can be sent without reference to a BASIC control program by patching some machine code into the operating system via the interrupt vector.

A further advantage of the servomotor is that only one data line is needed to send the control and refresh pulses, so only one bit of the computer's data port is committed to each servo

Commodore 64 Single Servo Control

The first part of the Commodore 64 source listing for a single servo control shows how the interrupt vectors (locations 788 and 789) are altered. This cannot be done using BASIC as an interrupt may occur during this alteration, causing the system to crash. Notice that the interrupts are turned off (SEI) while the alteration is made, and are re-enabled using CLI. The rest of the code is the interrupt handling routine for a single servo control.

The BASIC Calling program shows everything required to load the machine code routines, set up the user port, and then POKE values into memory location $3000 (12288) according to which key (1 to 9) is pressed. A motor connected to the user port should then move to a position proportional to the key value. Pressing E ends the session.

If you have an assembler, type in the source listing and assemble it into an object file that can be subsequently loaded by the BASIC Calling program. Alternatively, type in the BASIC Loader for the machine code and run this to load the code into memory. Type NEW before loading and running the BASIC Calling program. If you use the BASIC Loader, then lines 30 and 40 can be omitted.

Note: It is extremely important to note that if anything is wrong in a program that uses interrupts, the whole system can very easily become totally corrupted. This does not damage the computer, but you will probably have to switch the machine off and back on again to recover. Therefore, it is imperative to SAVE the program before RUNning it

Source Code

```
;+++++++++++++++++++++
;+++++++++++++++++++++
;++                 ++
;++ CBM SINGLE SERVO ++
;++      DRIVER     ++
;++                 ++
;+++++++++++++++++++++
;+++++++++++++++++++++
;
PORT = 56579 ; USER PORT DATA REG
ANGLE=12288  ; ANGLE VALUE LOCATION
;
*=$0334
;
      SEI         ;INTERRUPTS OFF
      LDA $0314   ;EXXISTING IRQ VECOTOR
      LDX $03C4
      STA $03C4
      STA $0314
      LDA $0315
      LDX $03C5
      STA $03C5
      STX $0315
;
      CLI         ;INTERRUPTS BACK ON
      RTS
;
;++++ EVENT HANDLER ++++
;
      PHP
      PHA
      TYA         ;SAVE REGISTERS
      PHA         ;ON STACK
      TXA
      PHA
      LDA #$FF
      STA PORT
      LDY #$FF
LOOP
      DEY         ;DELAY LOOP
      BNE LOOP    ;APPROX 1MSEC
;
      LDY ANGLE
LOOP1
      DEY         ;COUNT OUT PULSE
      BNE LOOP1
;
      LDA #$00
      STA PORT    ;ZERO DATA REGISTER
;
      PLA
      TAX         ;RESTORE REGISTER
      PLA         ;VALUES
      TAY
      PLA
      PLP
;
      JMP $EA31
```

BASIC Loader Program

```
10 REM **** BASIC LOADER FOR ***
*
20 REM **** SINGLE SERVO PROG***
*
30 :
40 FOR I=820 TO 882
50 READ A:POKE I,A
60 CC=CC+A
70 NEXT I
80 READ CS:IF CC<>CS THENPRINT
"CHECKSUM ERROR":STOP
100 DATA120,173,20,3,174,196,3,1
41,196
110 DATA3,141,20,3,173,21,3,174,
197,3
120 DATA141,197,3,142,21,3,88,96
,8,72
130 DATA152,72,138,72,169,255,14
1,3
140 DATA221,160,255,136,208,253,
172,0
150 DATA48,136,208,253,169,0,141
,3,221
160 DATA104,170,104,168,104,40,7
6,49
170 DATA234
180 DATA7170:REM*CHECKSUM*
```

BASIC Calling Program

```
10 REM **** SINGLE SERVO ****
20 :
30 DN=8:REM IF CASSETTE THEN
DN=1
40 IF A=0 THEN A=1:LOAD"SINGSERV
.HEX",8,1
50 POKE 964,79: POKE965,3:REM
POINT TO IRQ HANDLER
60 DDR=56577: POKE DDR,255:
REM ALL OUTPUT
70 MC=820:SYS MC: REM SET IRQ
VECTOR
80 POKE 53265,PEEK(53265)AND239:
REM BLANK SCREEN
90 :
100 GET K$:IF K$="" THEN100:REM
AWAIT KEYPRESS
110 REM ** ALTER MOTOR POSITION
**
120 IF ASC(K$)>48 AND ASC(K$)<58
 THEN POKE 12288,VAL(K$)*20
130 IF K$<>"E" THEN 80:REM 'E'
TO EXIT
140 END
```

SPATIAL AWARENESS

LOGO is a particularly useful language for investigating pattern and symmetry. We show you how a range of spatial transformations can be performed by the turtle, and develop a procedure that can alter other procedures, and will enable us to create strip patterns.

There are four kinds of transformation that we can apply to a two-dimensional figure and leave its shape unchanged (though its position may change). These transformations are: translation, rotation, reflection and glide reflection. Our diagram shows how a shape's position is changed by each of these transformations.

A figure is said to be *symmetrical* if we can transform it in one or more of these ways and leave its position, as well as its shape, unchanged. Finite shapes (such as polygons and letters of the alphabet) must have symmetries based on reflection and rotation, since translations and glide reflections will change their positions.

To investigate these symmetries it is useful to have LOGO procedures for reflecting and rotating shapes. We'll begin by looking at the task of reflecting a shape in a line that goes through the origin and has a given heading.

It is easiest if we assume that the procedure to draw the shape is state transparent (that is, it leaves the turtle in the same position with the same heading as it had before the procedure was run). Our task then breaks down into two parts: firstly, we need to find the co-ordinates and heading of the starting point of the reflection that corresponds to the starting point of the original shape. The second task needs to be performed before we start drawing the shape. It simply involves changing all right turns in the shape-drawing procedure to left turns, and all left turns to right turns. One way to do this is to replace all RTs and LTs in the procedure with a procedure called TURN, defined as follows:

```
TO TURN :A
   RT :DIR * :A
END
```

So we can now define a square as:

```
REPEAT 4 [FD 50 TURN 90]
```

To use this procedure we must first set the global variable DIR to 1. Thus MAKE "DIR 1 SQUARE will draw a square. To reflect the square in the y-axis all we need to do is type MAKE "DIR (–1) and then SQUARE. Try it and see what happens.

The procedure to position the turtle prior to drawing the reflection depends on a little bit of trigonometry:

```
TO REFLECT :A
   MAKE "H HEADING
   MAKE "XOLD XCOR
   MAKE "ANGLE ( ATAN :YOLD :XOLD ) – 90 + :A
   MAKE "R SQRT ( :XOLD * :XOLD + :YOLD * :YOLD )
   PU
   SETXY 0 0
   SETH :A + :ANGLE
   FD :R
   SETH 2 * :A – :H
   PD
   MAKE "DIR :DIR * (–1)
END
```

This procedure can now be used to see the effect of reflections in various lines through the origin. Try:

```
MAKE "DIR 1
PU SETXY 40 70 PD
SQUARE
REFLECT 60
SQUARE
```

If the reflected shape lies completely on top of the original then it is said to have 'reflective symmetry' about that line. Try:

```
MAKE "DIR 1
PU SETXY 0 0 PD
SQUARE
REFLECT 45
SQUARE
```

A similar procedure could be written to rotate a shape about a given point through a given angle, but we'll leave that for you to write.

Some patterns, such as those on wallpapers, use the same shape repeatedly in their design. It is possible to have translations and glide reflections that move the whole pattern, and yet leave it exactly as it was. For the moment, we'll concentrate on patterns involving translations along a single line, leaving two-dimensional patterns for the next instalment.

Combinations of the fundamental four transformations give rise to just seven kinds of patterns on a straight line. All these possibilities are shown in terms of a simple 'LEG' motif in our second diagram. We have built up procedures for drawing the seven patterns from any MOTIF using the procedures MOVE for translation, TURN for rotation, and R.MOTIF, which changes all the RT turns in MOTIF to LT turns, and all the LT turns to RT.

We have used LOGO's list processing facilities to write the procedure R.MOTIF by rewriting MOTIF. The procedure we use to do this is:

```
TO REWRITE :PROC
   OUTPUT REWRITE.PROC TEXT :PROC
END
```

REWRITE takes the text of a specified procedure, alters it and outputs it under another name. It assumes that the procedure it is working on is written in terms of LOGO primitives and does not contain any subprocedures. REWRITE contains a call to the following procedures:

```
TO REWRITE.PROC :TEXT
   IF :TEXT = [] THEN OUTPUT []
   OUTPUT FPUT REWRITE.LINE FIRST :TEXT
   REWRITE. PROC BUTFIRST :TEXT
END
```

This procedure divides the task of rewriting the input procedure into individual lines, by calling the following procedure:

```
TO REWRITE.LINE :LINE
   IF :LINE = [] THEN OUTPUT []
   IF LIST? FIRST :LINE THEN OUTPUT FPUT
   REWRITE.LINE FIRST : LINE REWRITE.LINE
   BUTFIRST :LINE
   OUTPUT FPUT
   CHANGE.WORD FIRST :LINE
   REWRITE.LINE BUTFIRST :LINE
END
```

REWRITE.LINE does the processing on each line, passing individual words on to CHANGE.WORD for it to deal with. The line beginning IF LIST? is needed in order to deal with a situation where MOTIF contains a REPEAT statement. If you exclude this possibility in your MOTIF procedures, then you can remove the line from this procedure. The listing for CHANGE.WORD is:

```
TO CHANGE.WORD :WORD
   IF ( ANYOF :WORD = "RT :WORD = "RIGHT ) THEN
   OUTPUT "LEFT
   IF (ANYOF :WORD = "LT :WORD = "LEFT ) THEN
   OUTPUT "RIGHT
   OUTPUT :WORD
END
```

This procedure checks each individual word and makes any necessary alterations. Having entered all these procedures, let's see how they work. First of all, we need to define a simple shape, such as:

```
TO TRI
   REPEAT 3 [FD 50 RT 120]
END
```

Now, enter DEFINE "REF REWRITE "TRI, and call up REF. Its definition should be:

```
TO REF
   REPEAT 3 [FD 50 LEFT 120]
END
```

It is quite possible to write a more general REWRITE procedure that will also rewrite any subprocedures called by the main procedure. If you should try to write this, take care with recursive procedures! You'll also need to be able to test whether a word is a procedure name.

THE SEVEN STRIP PATTERNS

It would be possible (and mathematically elegant) to build up the patterns from procedures for the four basic transformations. The pattern-drawing procedures make use of three helping subprocedures. These are:

```
TO POSITION
   HT
   PU
   SETXY – 125 0
   PD
END
```

This positions the turtle at the left-hand side of the screen, ready to begin drawing.

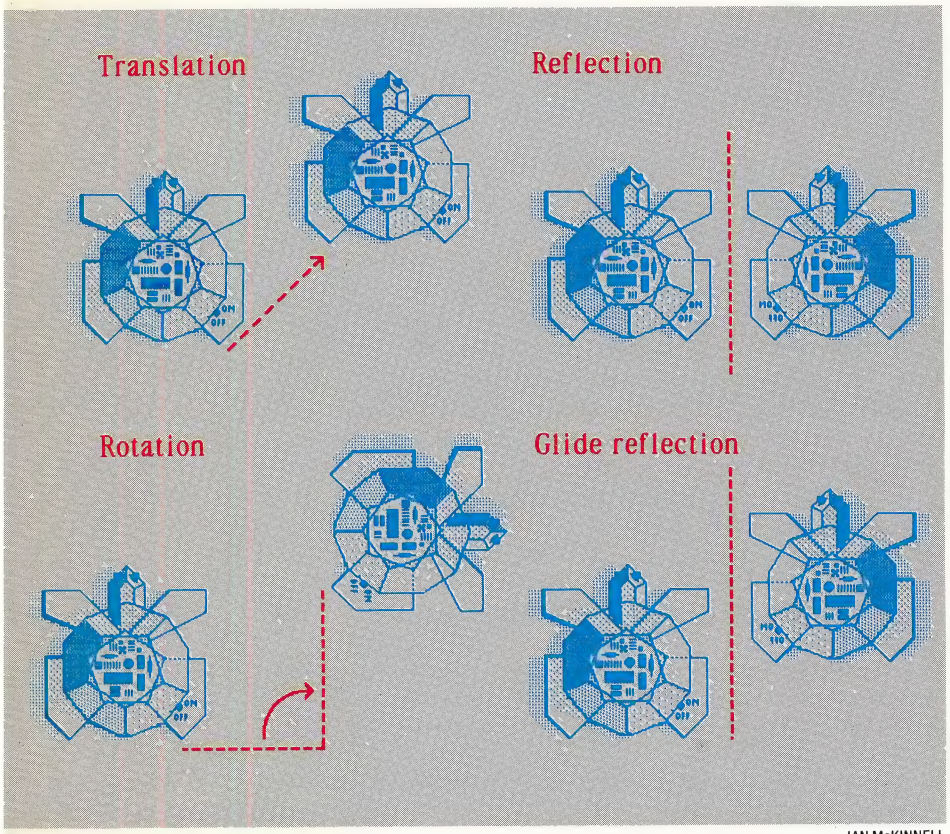
```
TO MOVE
   PU
   RT 90
   FD 50
   LT 90
   PD
END
```

MOVE performs the required translation.

```
TO TURN
   RT 180
END
```

TURN performs the one rotation that we require.

To use these procedures first define a shape procedure (say, SHAPE) which is state transparent and has no subprocedure calls. Then you can draw the first pattern using SHAPE as your motif by entering PATTERN1 "SHAPE.



The Isometry Cometh
Transformations that alter the position but not the shape of an object are known as isometries. Four basic types of isometric transformation exist: translation, rotation, reflection and glide reflection. Translation is a simple 'slide' of the original figure. Rotation turns the shape about some specified central point. Reflection involves the movement of points across a mirror line so that each point in the final shape lies the same distance on one side of the line as the corresponding point in the original did on the other. Glide reflection is a combination of a reflection and a translation. Whilst translation and rotation preserve 'sense', reflection and glide reflection change it: imagine reflecting a word in a mirror, for example

IAN McKINNELL

Seven Strip Patterns

To run the pattern procedures you must have the following procedures in the workspace: REWRITE.PROC, REWRITE.LINE, CHANGE.WORD, POSITION, MOVE and TURN. The seven possible patterns are:

```
TO PATTERN1 :PROC
   DEFINE "MOTIF TEXT :PROC
   POSITION
   REPEAT 6 [MOTIF MOVE]
END
TO PATTERN2 :PROC
   DEFINE "MOTIF TEXT :PROC
   DEFINE "R.MOTIF REWRITE.PROC TEXT
   :PROC
   POSITION
   REPEAT 3 [MOTIF MOVE TURN R.MOTIF
   TURN MOVE]
END
TO PATTERN3 :PROC
   DEFINE "MOTIF TEXT :PROC
   DEFINE "R.MOTIF REWRITE.PROC TEXT
   :PROC
   POSITION
   REPEAT 6 [MOTIF R.MOTIF MOVE]
END
TO PATTERN4 :PROC
   DEFINE "MOTIF TEXT :PROC
   DEFINE "R.MOTIF REWRITE.PROC TEXT
   :PROC
   POSITION
   REPEAT 6 [MOTIF TURN MOTIF TURN MOVE]
END
TO PATTERN5 :PROC
   DEFINE "MOTIF TEXT :PROC
   DEFINE "R.MOTIF REWRITE.PROC TEXT
   :PROC
   POSITION
   REPEAT 3 [MOTIF R.MOTIF MOVE TURN
   MOTIF R.MOTIF TURN MOVE]
END
TO PATTERN6 :PROC
   DEFINE "MOTIF TEXT :PROC
   DEFINE "R.MOTIF REWRITE.PROC TEXT
   :PROC
   POSITION
   REPEAT 6 [MOTIF TURN R.MOTIF TURN
   MOVE]
END
TO PATTERN7 :PROC
   DEFINE "MOTIF TEXT :PROC
   DEFINE "R.MOTIF REWRITE.PROC TEXT
   :PROC
   POSITION
   REPEAT 6 [MOTIF R.MOTIF TURN MOTIF
   R.MOTIF TURN MOVE]
END
```

These procedures need to be run with some suitable motif drawing procedure. The motif we used is:

```
TO LEG
   FD 50
   RT 90
   FD 20
   BK 20
   LT 90
   BK 50
END
```

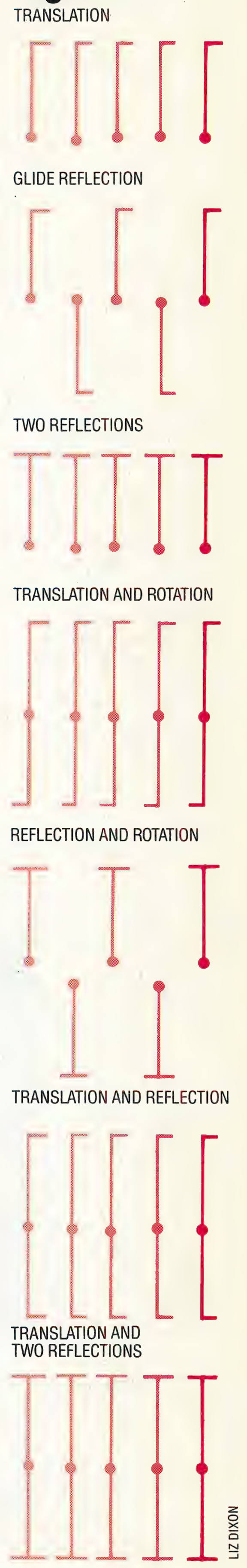
An alternative motif is:

```
TO FIG
   RT 30
   FD 20
   LT 50
   FD 20
   RT 90
   FD 10
   REPEAT 4 [FD 20 RT 90]
   BK 10
   LT 90
   BK 20
   RT 50
   BK 20
   LT 30
END
```

Logo Flavours

ATAN and TOWARDS do not exist in Atari LOGO, nor is there any simple replacement. This affects the REFLECT and ROTATE procedures, but not the PATTERN procedures.
Atari LOGO does not have TEXT and DEFINE as primitives, although the Atari manual does give a method of defining them. You could write R.MOTIF by simply modifying MOTIF using the editor

Logomotif



Seven Of A Kind
The four primitive isometric transformations may be combined in various ways to produce seven unique patterns, as shown here. In each case we start with the 'leg' motif, and all translations are made in the direction of the x-axis

CHECK MATE

One of the features of the Commodore 64 is the ease with which the location of screen data can be relocated in memory. In this part of the machine code course, we look at a routine that makes use of this facility to allow the design and storage of up to eight alternative screen displays.

Screen display and sprite handling are controlled by a special chip inside the Commodore 64 called the VIC II chip. The VIC chip accesses various sections of memory to obtain information from which it creates the display we see on the screen. These areas include the character ROM, where character shapes are held; the colour RAM, where colour information for the screen is held; and the screen RAM. The latter holds information about the characters to be displayed at any one of the 1,000 locations (25 rows by 40 columns) that go to make up the screen.

When the Commodore 64 is switched on, the VIC chip assumes that the screen is located in the 1,000 bytes starting at location 1024 ($0400), and it accesses this area to obtain its initial screen information. However, by altering the value of a register within the VIC chip, we can redirect the VIC chip to another area of memory — normally the first 16 Kbytes in memory. The upper four bits of the VIC control register at location 53272 ($D018) determine which one Kbyte block, out of the 16 Kbyte area in view, is interpreted as the screen. The following table shows the bit values that correspond to each of the 16 possible screen positions:

Bit Pattern	Start Of Screen	
0000XXXX	0	$0000
0001XXXX	1024	$0400 *
0010XXXX	2048	$0800
0011XXXX	3072	$0C00
0100XXXX	4096	$1000
0101XXXX	5120	$1400
0110XXXX	6144	$1800
0111XXXX	7168	$1C00
1000XXXX	8192	$2000
1001XXXX	9216	$2400
1010XXXX	10240	$2800
1011XXXX	11264	$2C00
1100XXXX	12288	$3000
1101XXXX	13312	$3400
1110XXXX	14336	$3800
1111XXXX	15360	$3C00

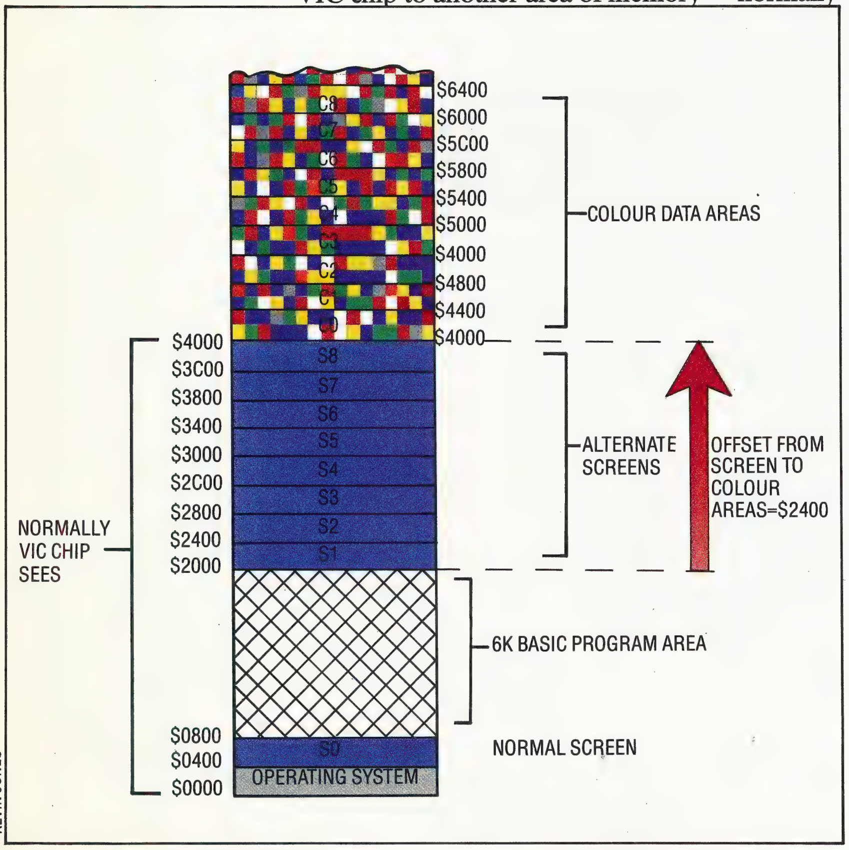
* = Default Position

To make the VIC chip move the screen to another area, we have to change the upper four bits of location 53272 ($D018) to the values shown in the table. However, we must not disturb the lower four bits (the XXXX part of the bit pattern in the table) as they control another function. To zero the upper four bits without changing the value of the lower four bits, we must AND the contents of the register with 15 (00001111 in binary). Having done this, we can then OR the new contents of the register with the value we require. To position the screen in the last area that the VIC chip can see — that is starting at 15360 ($3C00) — we would OR the contents of the register with 240 (11110000 in binary). In BASIC, the following POKE statement would do this:

```
POKE 53272,(PEEK(53272)AND15)OR240
```

Before we could write anything on our new screen we would also have to tell the Commodore 64's operating system that the position of the screen had changed. This is done by placing the HI-byte of the new screen's start address in location 648 ($0288). For the highest screen this is $3C, and is easily worked out in BASIC by dividing the screen start address by 256.

Having changed the contents of these two



VIC View
The Commodore 64's video controller (VIC) chip can 'see' 16K of memory. Normally this is the first 16K, $0000 to $3FFF, but it can be made to look at any of the other three 16K blocks by altering the contents of one of the VIC control registers. The 'Alternate Screens' program sets up nine alternate screen maps within the normal 16K area seen by the VIC chip. Corresponding colour maps for each screen are held in the RAM area just above the area seen by VIC, each screen except screen 0 having a constant offset of $2400 to its colour map

registers we can now use the new screen as normal. Note that if you wish to try changing the screen's position in BASIC, you will have to write and run a short program to do it.

We can use the Commodore 64's ability to move its screen to produce several interesting utilities. In particular, we can change the display quickly and with ease. The problem is that as we move the screen we also need to move the corresponding colour RAM, because the display will not make any sense unless the colour RAM contains the data appropriate to the screen being displayed. Although we can set up several different screen areas within memory and quickly change between them, there is only one unmoveable area of colour RAM, so to allow us to hold a number of separate screen displays we must set aside areas of memory to hold the 1,000 bytes of colour data for each screen. When we wish to display a screen we must copy this information into the colour RAM, and we must save this data to one of the areas of memory we have designated to hold colour RAM, before changing to a different screen.

ORGANISING MEMORY

The principle task of this utility is to organise memory for alternate screens (along with their corresponding colour data areas), and carry out the transfer of blocks of memory. As the VIC chip can 'see' 16 Kbytes of memory, we can design a system that allows us to have up to eight different screens and a sizeable BASIC program. The diagram shows the arrangement of memory used by the utility.

To ensure that none of the screen or colour areas is overwritten by a BASIC program, we must lower the top of BASIC memory. The following instruction in our BASIC calling program will perform this:

```
POKE 55,0:POKE 56,32:CLR
```

The base address of any screen can be calculated from its number by this formula:

Screen Base = \$1C00 + (\$0400*Screen Number)

The base address of the corresponding colour area can be calculated simply by adding an offset to the screen base address. The formula is:

Colour Base = \$2400 + Screen Base

The colour areas could be located anywhere in RAM, but it is most convenient to position them just above the last screen that can be seen by the VIC chip. Notice that a colour area for screen 0 — the normal screen — is included. For this particular colour area, the offset will be different and our program will have to take account of this.

The VIC control register and operating system register can be set for any screen by:

VIC register = \$70 + (\$10*Screen Number)
OS register = HI-byte of Screen Base Address

In addition to handling the setting of registers, and making the appropriate transfers of colour data to and from the colour RAM area, the program also stores the colour of the screen background and the border. These two features are controlled by a pair of registers in the VIC chip: 53280 (\$D020) controls the border colour and 53281 (\$D021) controls the screen background, or paper colour. The utility sets up a table within its own program area to store these two attributes for each screen.

MODES OF OPERATION

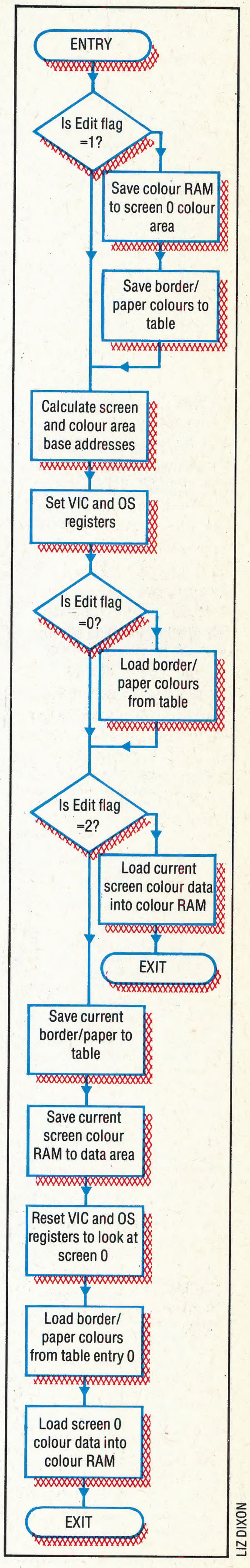
The utility has two modes of operation: it can either edit or display a selected screen. In each case, the screen number to be used must be POKEd into 49152 (\$C000). So that the same SYS call can be made, we use a special flag to indicate which mode has been selected. This flag is set by POKEing to location 49153 (\$C001).

0 – indicates display mode
1 – indicates edit mode

The edit mode works in an unusual way, in order to allow all the facilities of the screen editor (such as changing text colours, setting reverse mode and clearing the screen) to be used. The utility must first be called and the edit flag set to one. It will then save the normal screen colour area and the normal screen paper and border colours, set the VIC and operating system registers appropriately and then return to BASIC. At this stage, the BASIC calling program takes over, homing the cursor and then using the BASIC INPUT instruction. This instruction will wait for a return character (ASCII 13) before moving on. In the meantime, all the screen editor functions can be used in the normal way to edit the selected screen, pressing Return when each task is finished.

The BASIC program must then call the utility a second time, but on this occasion the edit flag is set to 2. This will save the colour data and paper/border colours of the screen you have worked on before restoring the normal screen. If you want to change the border or paper colour then the colours must be POKEd into locations 49154 (\$C002) and 49155 (\$C003) respectively. Although the routine is designed for this particular task, it incorporates general routines to set the control registers and copy data to or from the colour RAM. It would not, therefore, be a difficult task to produce a utility to your own specifications from these general routines.

The BASIC calling program is designed to display a menu giving the option to edit or display. The display routine calls up each of the eight different screens in sequence, in response to a keypress. The screens will continue to cycle round until the Return key is pressed. The program then restores the normal screen and returns to the menu. If the edit option is selected, the user can set the paper and border colours for the selected screen, and may then use the screen editor in the normal way to produce a picture. When you are finished, pressing Return will cause the picture to be stored and the normal screen to be restored.


ENTRY
Is Edit flag =1?
Save colour RAM to screen 0 colour area
Save border/paper colours to table
Calculate screen and colour area base addresses
Set VIC and OS registers
Is Edit flag =0?
Load border/paper colours from table
Is Edit flag =2?
Load current screen colour data into colour RAM
EXIT
Save current border/paper to table
Save current screen colour RAM to data area
Reset VIC and OS registers to look at screen 0
Load border/paper colours from table entry 0
Load screen 0 colour data into colour RAM
EXIT
LIZ DIXON

Basic Loader

```
10 REM ***********************
20 REM ** BASIC LOADER FOR   **
30 REM ** ALTERNATE SCREENS **
40 REM ***********************
50 :
60 FORI=49152 TO 49439
65 READA:POKE I,A
70 CC=CC+A
75 NEXT
80 READ CS:IFCS<>CC THENPRINT"CHECKSUM ERROR":STOP
100 DATA255,255,0,0,191,255,0,0,255
110 DATA191,0,0,191,191,0,0,255,191,0
120 DATA0,255,191,0,0,255,255,0,173,1
130 DATA192,201,1,208,30,169,0,141,6
140 DATA192,169,64,141,7,192,32,224
150 DATA192,162,0,32,246,192,173,2,192
160 DATA141,32,208,173,3,192,141,33
170 DATA208,173,0,192,208,6,32,32,193
180 DATA76,139,192,169,0,141,4,192,141
190 DATA5,192,174,0,192,173,5,192,24
200 DATA105,4,202,208,250,24,105,28
210 DATA141,5,192,24,105,36,141,7,192
220 DATA173,0,192,162,4,10,202,208,252
230 DATA24,105,112,141,8,192,173,24
240 DATA208,41,15,13,8,192,141,24,208
250 DATA173,5,192,141,136,2,173,1,192
260 DATA208,6,174,0,192,32,211,192,201
270 DATA2,240,4,32,189,192,96,174,0
280 DATA192,32,246,192,32,224,192,32
290 DATA32,193,169,0,141,6,192,169,64
300 DATA141,7,192,162,0,32,211,192,32
310 DATA189,192,96,173,6,192,133,251
320 DATA173,7,192,133,252,169,0,133
330 DATA253,169,216,133,254,32,3,193
340 DATA96,189,9,192,141,32,208,189,18
350 DATA192,141,33,208,96,173,6,192
360 DATA133,253,173,7,192,133,254,169
370 DATA0,133,251,169,216,133,252,32,3
380 DATA193,96,173,32,208,157,9,192
390 DATA173,33,208,157,18,192,96,162,3
400 DATA160,0,177,251,145,253,136,208
410 DATA249,230,252,230,254,202,48,10
420 DATA208,240,177,251,145,253,160
430 DATA231,208,232,96
440 DATA36606:REM*CHECKSUM*
```

Alternate Screens

```
3 REM  *****************************
5 REM  **                         **
10 REM **    ALTERNATE SCREENS    **
11 REM **                         **
12 REM *****************************
13 :
15 DN=8:REM FOR CASS DN=1
20 IFA=0 THEN A=1:LOAD"ALT SCREENS.HEX",DN,1
25 POKE55,0:POKE56,32:CLR:REM LOWER MEMTOP
40 SCNUMB=49152:   REM SCREEN NUMBER
70 EDITFG=49153:   REM 0=DISPLAY SCREEN
75 :               REM 1=EDIT SCREEN
77 :               REM 2=COPY C.RAM
80 ALT=49179:      REM M/C START ADDRESS
85 BRDCOL=49154:   REM BORDER COLOUR
87 PAPCOL=49155:   REM PAPER COLOUR
90 :
95 REM **** MAIN MENU ****
96 :
100 PRINTCHR$(147):REM CLEAR SCREEN
105 PRINTCHR$(154):REM LT BLUE LETTERS
110 DN$="■■":REM Q = CURSOR DOWN CHARS
130 PRINTTAB(8);DN$;"CBM 64 ALTERNATE SCREENS"
135 PRINTTAB(8);"------------------------"
140 PRINTTAB(5);DN$;"F1 - EDIT PICTURE"
160 PRINTTAB(5);DN$;"F3 - DISPLAY PICTURE SEQUENCE"
200 :
210 GETA$:IFA$=""THEN210:REM AWAIT KEYPRESS
220 IFA$="■" THEN GOSUB1000
230 IFA$="■" THEN GOSUB 1500
260 GOTO100
999 :
1000 REM **** EDIT SCREEN ****
1002 :
1005 EF=1:REM SET EDIT MODE
1010 PRINTCHR$(147)
1020 PRINTTAB(16);DN$;"EDIT MODE"
1030 PRINTDN$;:INPUT"SCREEN NUMBER";SN$
1050 IFASC(SN$)<48 OR ASC(SN$)>56 THEN 1030
1060 PRINTDN$;:INPUT"BORDER COLOUR";BC$
1070 IFVAL(BC$)=0 AND BC$<>"0"THEN 1060
1080 PRINTDN$;:INPUT"PAPER COLOUR";PC$
1090 IFVAL(PC$)=0 AND PC$<>"0"THEN 1080
1100 :
1110 POKEEDITFG,EF
1120 POKESCNUMB,VAL(SN$)
1130 POKEBRDCOL,VAL(BC$)
1140 POKEPAPCOL,VAL(PC$)
1150 SYS ALT
1155 REM ** WAIT FOR RETURN **
1162 INPUT"■";X$:   REM S =HOME CURSOR
1165 REM ** SAVE SCREEN **
1170 EF=2
1175 POKE EDITFG,EF
1180 SYS ALT
1185 RETURN
1190 :
1500 REM **** DISPLAY SCREEN ****
1510 EF=0
1520 PRINT CHR$(147)
1545 SN=1
1550 POKE EDITFG,EF
1555 POKE SCNUMB,SN
1560 SYS ALT
1570 GET X$:IFX$=""THEN1570:REM AWAIT KEYPRESS
1572 IFX$=CHR$(13)THEN 1580:REM NORMAL SCREEN
1575 SN=SN+1:IFSN<9 THEN 1555
1577 SN=1:GOTO1555
1580 POKESCNUMB,0:SYS ALT
1600 RETURN
```

Commodore 64

```
;++++++++++++++++++++++
;++++++++++++++++++++++
;++                  ++
;++    ALTERNATE     ++
;++     SCREENS      ++
;++       FOR        ++
;++      CBM  64     ++
;++                  ++
;++++++++++++++++++++++
;++++++++++++++++++++++
;
FROM    =$FB                ;0 PAGE
TO      =$FD                ;POINTERS
;
CRAMLO=$00                  ;START OF COLOUR RAM
CRAMHI=$D8
NCOLLO=$00                  ;NORMAL COLOUR
NCOLHI=$40                  ;BASE ADDRESS
NSCRHI=$04                  ;NORMAL SCREEN BASE ADDRESS
NVCPOK=$10                  ;NORMAL VIC REG VALUE
;
BLOCKS=$03                  ;NO OF 256 BYTE BLOCKS
EXTRA  =$E7                 ;EXTRA AMOUNT TO 1000 BYTES
;
COLOFF=$24                  ;COLOUR OFFSET HIBYTE
SCROFF=$1C                  ;SCREEN OFFSET HIBYTE
VICOFF=$70                  ;VIC CTRL REG OFFSET
VCMASK=$0F                  ;VIC CTRL REG LOBYTE MASK
VICREG=$D018                ;SCREEN LOCATION CONTROL REG
EDREG=$0288                 ;SCREEN EDITOR KERNAL REG
BORDER=$D020                ;BORDER COLOUR REG
PAPER  =$D021               ;BACKGROUND COLOUR REG
;
*=$C000                     ;SET LOAD POINTER
;
SCNUMB *=*+1                ;SCREEN NUMBER
EDITFG *=*+1                ;EDIT MODE FLAG
BRDCOL *=*+1                ;BORDER COLOUR
PAPCOL *=*+1                ;PAPER COLOUR
;
SCBASE *=*+2                ;SCREEN BASE STORAGE
CLBASE *=*+2                ;COLOUR BASE STORAGE
VPOKE  *=*+1                ;TEMP STORE FOR VIC NUMBER
BRDTAB *=*+9                ;BORDER COLOURS TABLE
PAPTAB *=*+9                ;PAPER COLOURS TABLE
;
;++++ SAVE SCREEN 0 ++++
        LDA EDITFG
        CMP #$01            ;IF 0 OR 2
        BNE CALC            ;THEN DON'T SAVE
;
        LDA #NCOLLO
        STA CLBASE
        LDA #NCOLHI
        STA CLBASE+1
        JSR SAVE            ;SAVE CRAM TO C0
;
        LDX #$00
        JSR SAVEBP          ;SAVE B/P REGS
;
        LDA BRDCOL
        STA BORDER          ;SET NEW B/P
        LDA PAPCOL          ;COLOURS
        STA PAPER
;
;++++ CALCULATE COLOUR BASE ++++
;
CALC
        LDA SCNUMB
        BNE NOZERO
        JSR RESET           ;SET NORMAL REGS
        JMP TESTFG
;
NOZERO
        LDA #$00
        STA SCBASE
        STA SCBASE+1        ;INIT SCBASE
;
        LDX SCNUMB          ;LOAD SCREEN NUMBER
        LDA SCBASE+1        ;HI BYTE ONLY
MULT
        CLC
        ADC #$04
        DEX
        BNE MULT
;
        CLC
        ADC #SCROFF         ;ADD SCREEN OFFSET
        STA SCBASE+1
;
        CLC
        ADC #COLOFF         ;ADD COLOUR OFFSET
        STA CLBASE+1
;
;++++ SET VIC AND EDITOR REGISTERS ++++
        LDA SCNUMB
        LDX #$04
MORE
        ASL A
        DEX
        BNE MORE            ;MULT BY 16
;
        CLC
        ADC #VICOFF         ;ADD OFFSET
        STA VPOKE
        LDA VICREG
        AND #VCMASK
        ORA VPOKE
        STA VICREG          ;SET VIC REG
;
        LDA SCBASE+1
        STA EDREG           ;SET EDITOR REG
;
;++++ TEST STATUS OF EDIT FLAG ++++
TESTFG
        LDA EDITFG
        BNE NOLOAD          ;IF 0 DISPLAY MODE
        LDX SCNUMB
        JSR LOADBP          ;LOAD B/P TO REGS
;
;
NOLOAD
        CMP #$02
        BEQ CONT            ;IF 2 THEN SAVE RAM
;
        JSR LOAD            ;LOAD COL TO CRAM
        RTS
;
CONT
        LDX SCNUMB
        JSR SAVEBP          ;SAVE BP REGS
        JSR SAVE            ;SAVE CRAM TO COL
        JSR RESET           ;SET NORMAL REGS
        LDA #NCOLLO
        STA CLBASE          ;LOAD CBASE WITH C0 BASE
        LDA #NCOLHI
        STA CLBASE+1
        LDX #$00
        JSR LOADBP          ;RELOAD ORIG BORD/PAP COLS
        JSR LOAD            ;LOAD NRML SCR COLS
        RTS
;
;+++++ TRANSFER TO RAM S/R ++++
LOAD
        LDA CLBASE
        STA FROM            ;LOAD 0 PAGE PTRS
        LDA CLBASE+1
        STA FROM+1
;
        LDA #CRAMLO
        STA TO
        LDA #CRAMHI
        STA TO+1
;
        JSR COPY            ;COPY RAM AREA
        RTS
;
;++++ LOAD BORDER & PAPER COLS S/R ++++
LOADBP
        LDA BRDTAB,X
        STA BORDER
        LDA PAPTAB,X
        STA PAPER
        RTS
;
;++++ TRANSFER FROM RAM S/R ++++
SAVE
        LDA CLBASE
        STA TO              ;LOAD 0 PAGE PTRS
        LDA CLBASE+1
        STA TO+1
;
        LDA #CRAMLO
        STA FROM
        LDA #CRAMHI
        STA FROM+1
;
        JSR COPY            ;COPY RAM AREA
        RTS
;
;++++ SAVE BORDER & PAPER COLS S/R ++++
SAVEBP
        LDA BORDER
        STA BRDTAB,X
        LDA PAPER
        STA PAPTAB,X
        RTS
;
;++++ COPY 1000 BYTES S/R ++++
;
COPY
        LDX #BLOCKS
        LDY #$00
NEXT
        LDA (FROM),Y
        STA (TO),Y
        DEY
        BNE NEXT
NXBLOC  INC FROM+1
        INC TO+1
        DEX
        BMI FINISH
        BNE NEXT
        LDA (FROM),Y
        STA (TO),Y
        LDY #EXTRA          ;EXTRA BYTES
        BNE NEXT
FINISH
        RTS
;
;++++ RESET VIC AND EDIT REGS S/R ++++
RESET
        LDA #NCOLLO
        STA CLBASE
        LDA #NCOLHI
        STA CLBASE+1
        LDA VICREG
        AND #VCMASK
        ORA #NVCPOK
        STA VICREG          ;RESTORE VIC REG
        LDA #NSCRHI
        STA EDREG           ;RESTORE EDREG
        RTS
```

NOW IS THE TIME TO ORDER ANY COPIES OR BINDERS YOU MAY BE MISSING FROM YOUR COLLECTION.

Copies of any weekly issue can be obtained, subject to the availability of stocks, by using this reply-paid order form and marking clearly which issues you require to be sent to you.

Each issue costs 80 pence including postage and packing, and please enclose your cheque/postal order made payable to Orbis Publishing Limited.

Back Numbers Order Form.

Please send me the back numbers I have circled below.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43					

Each issue costs 80 pence including postage and packing.

Binder Order Form.

Please send me the binder volume numbers I have ticked below.

☐ Volume 1 ☐ Volume 2
☐ Volume 3 ☐ Volume 4

Binders are £3.95 each inclusive of postage and packing.

Important: Please read this carefully:
1. Do not complete this order form if you have already asked for binders to be sent to you automatically as they are issued.
2. Readers not in the UK or The Republic of Ireland, see inside front cover for details of how to obtain binders and back numbers.

I enclose a cheque/postal order made payable to: Orbis Publishing Ltd, for a total of £________ which I understand includes the cost of postage and packing.

NB: Please allow 28 days for the delivery of both back numbers and binders.

When you have completed the order form above for back numbers and/or binders, fill in your name and address in the space provided.
Then cut along the dotted line to detach the page, enclose your cheque/postal order, and fold the page carefully.
NO STAMP NECESSARY.

BLOCK CAPITALS PLEASE

Complete the section below with one letter, figure or space per square.

Mr
Mrs
Miss

Initials

Surname

Address

Send this page with your cheque/postal order to:
The Home Computer Advanced Course,
Orbis Publishing Limited,
Freepost
Orbis House, 20-22 Bedfordbury,
London WC2N 4BR.

(No stamp necessary)

© 1983 LISTER, D.—OHIO STATE UNIV.